SYBASE®

An SAP® Company

Users Guide

Sybase IQ InfoPrimer 15.3

Microsoft Windows, Linux, and UNIX

DOCUMENT ID: DC01640-01-1530-01
LAST REVISED: May 2011

Contents

About Sybase IQ InfoPrimer

Sybase® IQ InfoPrimer provides a mechanism for extracting data from various datasources, transforming and cleaning data, and then loading it into Sybase IQ. It also lets you extract and load data from datasources into Sybase IQ, then perform a set of transformations on the data directly in the Sybase IQ target database.

Sybase IQ InfoPrimer capabilities include:

- Data extraction – extract data from various datasources, including Adaptive Server® Enterprise, IBM DB2, Oracle, Microsoft SQL Server, SQL Anywhere®, Sybase IQ, MySQL, text files, and XML files.
- Data transformation – convert, clean, merge, and split data streams as well as transform data already loaded into Sybase IQ.
- Data loading – load data into an IQ target database using update, insert, delete, or bulk copy statements.

Sybase IQ InfoPrimer Architecture

Sybase IQ InfoPrimer includes Sybase IQ InfoPrimer Development and Sybase IQ InfoPrimer Server.

- IQ InfoPrimer Development – which is available only on Windows, is a graphical user interface (GUI) tool for creating and designing data transformation projects and jobs.

  You can create a data transformation flow using any or all project types: Extract and Load (EL), SQL Transformation, and Extract Transform and Load (ETL).
- IQ InfoPrimer Server – is a scalable and distributed grid engine that connects to datasources, and extracts and loads data to data targets using transformation flows designed using IQ InfoPrimer Development.

  Note: The terms grid engine and IQ InfoPrimer server are used interchangeably in this guide.

By default, IQ InfoPrimer Development includes an embedded IQ InfoPrimer Server, which is a runtime engine that controls processing, such as connecting to databases and executing procedures. However, to perform parallel execution of jobs and projects, you must install Sybase IQ InfoPrimer Server, which is available as a separate executable.

You can add multiple servers on different operating systems within your network. All the Sybase IQ InfoPrimer Servers that you add to the grid must be registered. You can use the Engine Manager, available within IQ InfoPrimer Development interface to register servers.

Each server exposes certain services to all other peer servers. IQ InfoPrimer uses the various servers on a grid for parallel execution of projects and jobs.

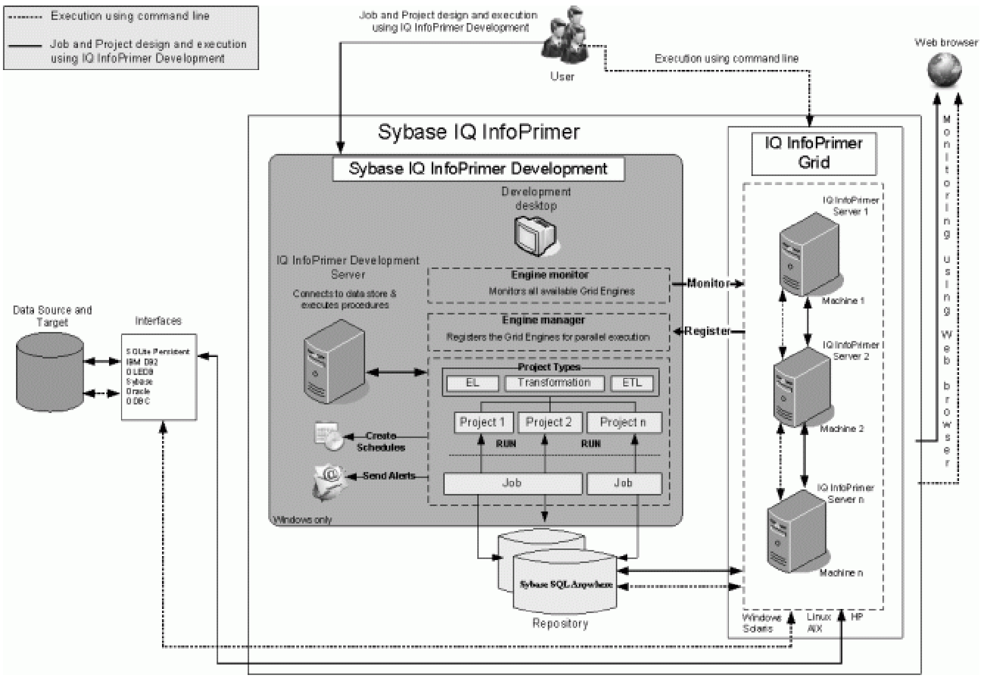
IQ InfoPrimer Server connects to the destination or the source database using methods or drivers called interfaces. One of the supported interfaces, the Sybase SQL Anywhere 11 ODBC driver, which is used to connect to Sybase SQL Anywhere, is automatically installed by the IQ InfoPrimer Development installer. Sybase IQ ODBC driver, which is used to connect to Sybase IQ is packaged with the Sybase IQ 15.3 Server or Client. To install the other supported interfaces, see the respective vendor documentation.

Note: Sybase SQL Anywhere 11 is packaged with the IQ InfoPrimer Server and must be installed manually.

You can monitor the Sybase IQ InfoPrimer Servers installed in your network using either the Engine Monitor or a Web browser.

You can use the Sybase IQ InfoPrimer Development interface to create schedules and alerts for projects and jobs.

Figure 1: Sybase IQ InfoPrimer Architecture



Repositories

Repositories contain all data and information related to Sybase IQ InfoPrimer objects, projects, and jobs.

Repositories are held in a collection of data store tables in a SQL Anywhere database. Sybase IQ InfoPrimer allows using multiple data stores as well as multiple separate repositories in a single data store (using the same tables).

A repository is identified by the combination of data store and client. Clients provide a logical separation of repositories in the same data store tables, the names can be freely chosen.

Each client can support any number of users; each user has a user name and password that controls access to the information.

Note: Do not manually manipulate data in the repository. Sybase cannot guarantee the functionality of a repository after it has been manually manipulated.

Multiple repositories are simultaneously accessible during a session. You can copy and transfer projects and jobs between repositories, which allows you to separate your production repository from your development repository.

Project Types

Sybase IQ InfoPrimer Development offers three different project types for creating a data transformation flow.

- The Extract Load (EL) Project – extract data from any supported datasource and load the data into a Sybase IQ database.
- SQL Transformation Project – create a transformation project that models the data transformation and generates executable transformation SQL for Sybase IQ. This project type is specialized to perform SQL transformations where the source and target are both on the same instance of Sybase IQ.
- Extract Transform and Load (ETL) Project – extract data from multiple heterogeneous datasources, apply a series of transformation rules or functions to the extracted data to derive the data for loading into the end target, and load it into one or more Sybase IQ data targets.

  ETL and SQL Transformation projects are both collections of components, links, and transformation rules. Each project contains one or more steps that are executed sequentially when the project is run. ETL and SQL Transformation projects consist of various components and links, which connect components through their ports. These components are stored in the Component Store and each of them is dedicated to a specific task, and incorporates task-specific features. The ETL project components connect to various datasources from which they read data to transform based on the transformation rules.

Schedules

Use the Runtime Manager in the Sybase IQ InfoPrimer Development interface to manage projects and jobs and see an overview of your current scheduled task.

Use the Runtime Manager to create, edit, delete, execute, list, import, and terminate scheduled tasks using a selected engine. You can schedule the start and end date and time for the execution of projects or jobs. You can also specify the frequency of project or job execution.

Alerts

Set alerts to be notified by e-mail messaging of runtime events, such as project or job execution, or when an error message occurs.

Use the Alert Manager from the IQ InfoPrimer Development interface to create, edit, and delete alert definitions on the engine you are connected to.

You can also use the File Log Inspector or the IQ InfoPrimer Web interface to view event alert history. Alert history is stored in a separate log file for help during troubleshooting. You can also set filter conditions while mapping an alert.

Monitor

Sybase IQ InfoPrimer lets you monitor grid engines, projects, and jobs.

You can use a Web browser to:

- Monitor the state of all grid nodes, including the IQ InfoPrimer Development grid engine that is a part of the grid architecture.
- Monitor project and jobs that are launched from IQ InfoPrimer Development interface.
- Monitor the state of a remote job.
- Suspend and resume a remote job and project.
- View the remote log file of an IQ InfoPrimer Server.
- View alert history.
- View the list of scheduled tasks.

You can also use the Engine Monitor in the IQ InfoPrimer Development interface to monitor the grid engines installed on your network.

Getting Started

Become familiar with the Sybase IQ InfoPrimer Development interface, its various components, and learn how to make customized groups of settings using the interface.

Starting Sybase IQ InfoPrimer

Launch the Sybase IQ InfoPrimer Development interface from the Windows Start menu.

1. In Windows, select **Start > Programs > Sybase > Sybase IQ InfoPrimer Development 15.3 > Sybase IQ InfoPrimer Development**.
2. In the Repository Logon window, create a new repository connection. See Configuring a Repository Database and Adding and Editing a Repository Connection.

   Click **Logon.**
3. Select the task you want to perform either from the Quick Start options or using the Navigator. To create a new project, job, or template from the Navigator, right-click any of the project type folder and select **New >** ***Project Type*****, Job, or, Template**.

See also

- *Adding and Editing a Repository Connection* on page 14
- *Configuring a Repository Database* on page 13

Using the Sybase IQ InfoPrimer Development Interface

The Sybase IQ InfoPrimer Development interface includes various components including the Design window, Navigator, Properties window, Component store, and the Assistant window.

Design Window

Use the Design window to create, modify, simulate (valid only for ETL project type), and execute projects and jobs.

When you start Sybase IQ InfoPrimer Development, the design window shows quick-start options for some tasks. Alternatively, you can also perform these tasks from the Navigator.

If you do not want these to show on start-up, disable them by selecting **File > Preference** and unselecting **Show Quick Start**.

Adding Components to the Design Window for Projects and Jobs

For a SQL Transformation project, an ETL project, or a job, use the Design window to add and connect components and set their properties.

Add components to projects and jobs in any of these ways:

- Select the component in the Component Store and drag it to the Design window.
- Double-click a component in the Component Store.
- Right-click a component in the Component Store and select **Add.**

  You can also add components to projects by right-clicking the port of an existing component in the Design window and selecting **Add Component**. Select the component type and then the component you want to add. The component is added before or after the existing component, depending on whether the selected port is an input port or an output port.

Deleting Components

Remove components from projects using the Design window.

1. In the Design window, select the component to delete.
2. Right-click and select **Delete.**

Processing General Commands

Use the general project menu to process commands such as Close and Print.

1. Right-click anywhere in the Design window to open the general project menu.
2. Right-click a component and select the action to perform.

Properties Window

The Properties window lets you view and edit the properties that apply to components, projects, or jobs.

Note: For EL projects, the Properties windows only displays advanced project properties, if there are any. Property options such as externalization, encryption, and evaluation can be set using the Advanced tab in the EL configuration window.

For the ETL and SQL Transformation projects, use the Properties window to perform these tasks:

View and edit component properties

To view and edit component properties and values, select the component in the Design window. All properties of the selected component appear in the Properties window.

Identify mandatory properties

A property name displayed in bold text in the Properties window indicates that the property is required for the component to operate correctly.

All other properties are optional; use them to fine-tune and configure the component.

Allow dynamic values

The Evaluate option lets you compute and evaluate dynamic property settings at runtime instead of assigning static values at design time.

Select the **Evaluate** option to allow expressions within component property values. For some property items, this option is by default, selected.

Use the Eval check box in the Properties window to enable or disable evaluation for a property. Once the Evaluate option is activated, you can include JavaScript expressions in the corresponding field using square bracket notation (SBN).

Encrypt properties

Project and job data, as well as property values, are stored in the Sybase IQ InfoPrimer repository. Most of the records in the repository are not encrypted.

By default, the Encrypt option is selected for the password property.

To encrypt property values, right-click a property name in the Properties window, and select **Encrypt.**

Alternatively, select the **Encrypt** check box next to the property value.

Add and edit custom properties

You can add or edit custom component properties in the Properties window. Like other properties, custom properties also incorporate a variable that can be referenced in expressions or user-defined procedures.

Access the component configuration window

Click the **Property** icon in the Properties window to access the configuration window of a selected component.

Enable transactionality for projects and jobs

Select the **Propagate rollback** option in the Properties window to enable transactionality support for generated jobs or projects.

Note: Transactionality of an EL project can be set in the Advanced tab of the EL configuration window.

When you select this option, data is committed at the end of the write operation for a successful execution, and rolled back for an unsuccessful execution. If you do not select the option, the project does not enforce a transaction rollback on successful components, if one or more components fail. Failed components roll back their own transactions. Jobs also, when Propagate rollback is not selected, do not enforce transaction rollbacks on successful projects, if one or more projects fail. Failed projects roll back their own transactions.

Note: If Propagate rollback is selected for a project or job, and there are multiple components that are a target for the same table, also select the Shared connection property for all these components. Otherwise, Sybase IQ InfoPrimer may stop responding.

See also

- *Configuring an EL Project* on page 33

Navigator

The Navigator lets you browse the repository and administer projects, jobs, and user accounts.

Use the Navigator to:

- Navigate and browse the repository
- Administer the repository.
- Administer projects and jobs
- Administer user accounts.
- Execute projects and jobs.

Navigating and Browsing a Repository

The Navigator allows you to navigate and browse a repository.

In the Navigator, the hierarchical tree list represents:

- Open repositories
- Client user sessions to the open repositories
- Objects stored in the repository, such as projects, jobs, and templates
- Recently opened projects, jobs, and templates

A repository can be simultaneously opened by multiple client user sessions. A client user is part of a client. Both client users and clients are registered when they log on to the repository.

The following example shows the tree structure:

```
Repositories
-- <RepositoryName1>
---- <ClientUser1>.<Client1>.<Repository Name1>
------ Recent
-------- Recently opened projects
-------- Recently opened jobs
-------- Recently opened templates
------ ETL Projects
-------- Project_1
-------- Project_2
-------- Project_N
------ Extract and Load Projects
-------- Project_1
```

```
-------- Project_2
-------- Project_N
------ SQL Transformation Projects
-------- Project_1
-------- Project_2
-------- Project_N
------ Jobs
-------- Job_1
-------- Job_2
-------- Job_N
------ Templates
-------- Template_1
-------- Template_N
---- <ClientUser1>.<ClientM>.<Repository Name1>
---- <ClientUserN>.<Client1>.<Repository Name1>
-- <RepositoryName2>
```

Sorting Projects and Jobs Within Folders

The Navigator allows you to change the sort order of projects and jobs in folders.

1. Click the **Select Navigator Display Preference** icon in the top-right corner of the Navigator.
2. Select **Sort Order** and then one of these options: **Alphabetical**, **Creation Date**, and **Modification Date**.

Defining the Presentation of Projects

The Navigator lets you to define how you want the projects to be displayed.

1. Click the **Select Navigator Display Preference** icon in the top-right corner of the Navigator.
2. Select **Presentation** and then one of the options, namely Flat, or Hierarchical. A flat presentation displays all project types under a common project folder. A hierarchical presentation, which is the default, displays each project under the respective project type folder name.

Component Store

Use the Component Store to add components to a SQL Transformation project, ETL project, or a job.

- Drag the component onto the Design window.
- Right-click a component and select **Add.**
- Double-click a component.

Setting User Preferences

Use the Preferences window to customize groups of settings in Sybase IQ InfoPrimer Development.

1. From the main Sybase IQ InfoPrimer Development interface, select **File > Preferences**.
2. Select **Appearance**, and set these options:

   - Locale for user interface display – select the locale language for your environment: **_de** (German), **_en_US** (US English, the default), or **_en_GB** (UK English).
   - Show assistant for creating projects – view the assistant that guides you through completing a project, and displays information on the current state of the open project.
   - Default font for displaying text – select the font for displaying text file contents in the Text Data Provider and Text Data Sink component windows. This setting is useful when you work with non-Western character sets, such as Unicode. The default font is Monospaced. The recommended fonts for displaying text are Dialog or Monospaced.
   - Default font for displaying data – select the font for displaying port data throughout the application. This setting is useful when you work with non-Western character sets, such as Unicode. If you are installing Sybase IQ InfoPrimer Development for the first time, the default font is Tahoma. If you have previously installed this version of Sybase IQ InfoPrimer Development, the font is set to the previously defined value. Sybase recommends that you set the font to Tahoma.

     Note: To use “Default font for displaying text” and “Default font for displaying data” options, Sybase recommends that you install the files for East Asian languages. In the Windows Control Panel, click the Regional and Language Options, select the **Languages** tab, then select **Install Files for East Asian Languages**. Enabling this option installs the required fonts for displaying Unicode characters.
   - Create links automatically when components are added – automatically create links between an existing component and new components added to the project.
   - Default action on double-clicking a connection (ETL project type only) – specify whether to open the Mapping window (default) or the Preview window when you double-click a connection during simulation. The Mapping window displays the mapping information and the Preview window provides a preview of the connection data.
   - Display qualified transformation objects – add a prefix consisting of the owner name to the name of the objects in the Navigator. For example, if this option is selected, a project name appears in the Navigator as:

     ```
     TRANSFORMER.Demo Character Mapper
     ```

     where TRANSFORMER is the name of the client user who created the project.

- Use unique object names – enforce unique project and job names on a repository connection.
- Show password in component tooltips – display in component tooltips, the password used to log in to the underlying database. By default, the password appears in the tooltips as a series of asterisks.
- Use enhanced color accessibility – change the color of the component ports to enhance support for users with color disabilities. Selecting this option changes the default color of the port from yellow to blue, enabling users with color disabilities to distinguish between port states.
- Use vertical component layout – display the alignment of projects and jobs vertically instead of left to right, which is the default.
- Show information dialogs – perform actions without being interrupted by information prompts, unselect this option.
- Show Quick Start – display the Quick Start pane on start-up.
- Number of recently opened projects, jobs, and templates to display – specify the number of recently accessed objects to view in the Navigator and the File menu.
- Open the last repository automatically – connect automatically to the last logged on repository, on restart.
- Save repository client password – save the client password after you log in to the repository. If selected, you do not have to provide the password during your next login.

3. Select **Data Viewer** and specify the maximum length of an exported data field. The default value is 255. Data fields longer than the value you specify here are truncated in the export files.
4. Select **Query Designer** and set these options:

   - Enable delete functionality of database objects – delete all records of a selected table when you right-click the table in the Query Designer and select **Truncate Object**.
   - Default number of records to retrieve from the Query Designer. The default is 25.
   - Create joins automatically – automatically create joins between identical attribute names used within tables or views.
   - Use brackets when creating joins – automatically surround join clauses with brackets in the generated query. For example, the **select** statement would appear as:

     ```
     select * FROM SALES ((<join statement 1>)
     <join statement 2>)
     ```
   - Default number of recently accessed tables and views to display – specify the number of recently accessed tables and views you want the Query Designer to display in the Recent tab. The default value is 25.
5. Select **Engine** and set these options:

   - Start local engine during application start-up.
   - Interval between engine monitor updates (sec)– specify the number of seconds to wait between updates of the Engine Monitor. The default is 5 seconds.

- Rate of simulation (msec) – set simulation trace delay. The simulation trace delay option accepts values between 10 and 9999 milliseconds. The default is 250 milliseconds.
- Grid engine ping timeout (sec) – specify the number of seconds allowed for accessing the grid engine before starting or restarting a local grid engine. The default is 60 seconds.
- Interval between progress monitor updates (sec) – specify the number of seconds to wait between updates of the progress monitor for a job execution. The default is 5 seconds.
- Allow selection of the execution engine – specify the grid engine to be used for project execution.
- Execution engine server – specify the IP address of the primary grid engine server.
- Execution engine port – specify the port address of the primary grid engine server.

6. Select **Performance Logging** and specify the detail level for logging performance data:
   - 0 – performance data is not written to the repository.
   - 1 – performance data is written to the repository. This is the default value.
7. Click **Save**. For some of the changes to take effect, you may be prompted to restart Sybase IQ InfoPrimer Development. If you select not to restart when prompted, the changes take effect the next time you start the application.

Administration Tasks

Review the tasks a Sybase IQ InfoPrimer administrator needs to perform.

Creating and Administering Repositories

Repositories contain all data and information related to Sybase IQ InfoPrimer objects, projects, and jobs.

Configuring a Repository Database

Access and define the ODBC datasource for the empty SQL Anywhere 11 repository.

Repositories are held in a collection of tables in a SQL Anywhere database. Sybase IQ InfoPrimer allows physical separation of repositories by using multiple repository databases.

1. Navigate to `<`*`IQIPD_install_directory`*`>\etc`, copy `repositoryEmpty.db` and rename to a name of your choice. For example, `myRepository_IQIP153.db.`
2. Configure the ODBC datasource names to define your own repository by creating a System datasource that points to `myRepository_IQIP153.db`.
    1. Select **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

       Note: To administer datasources on a 64-bit machine, run `odbcad32.exe` from `C:\WINDOWS\SysWOW64`.
    2. Click **Add**.
    3. Select **SQL Anywhere** and click **Finish**.
    4. Click the **ODBC** tab. In the Data source name field, enter `myRepository_IQIP153`.
    5. Click the **Login** tab and enter `dba` as the user ID and `sql` as the password.
    6. Click the **Database** tab and enter these values:
        - A server name.
        - Start line – `C:\Sybase\IQIPD-15_3\dbeng11.exe.` This is the default installation location.

          Note: IQ InfoPrimer Development runs `dbeng11.exe` for configuring the IQ InfoPrimer repository. `dbeng11.exe` is a single-user database that cannot be accessed from other machines. To access the repository from another machine, specify `dbsrv11.exe` instead of `dbeng11.exe` in the Start line field.

- Database file – `C:\Sybase\IQIPD-15_3\etc\myRepository_IQIP153.db`.

7. Click the **ODBC** tab and click **Test Connection** to verify the connection.
8. Click **OK**.

Opening a Repository Connection

To open a repository, you must assign at least one client and one client user. A client can have multiple users.

1. Select **File > Open Repository**. Or, in the Navigator, right-click **Repositories** and select **Open Repository**.
2. Select a repository from the Connection list and click **Logon.**

Closing a Repository Connection

Closing a repository ends all user sessions currently connected to the repository.

1. In the Navigator, right-click the repository name and select **Close Connection**.
2. Click **Yes**.

Note: If there are no open projects or jobs, the repository closes automatically without confirmation.

Closing a Client User Session

Closing a client user session closes the client and all open projects and jobs.

1. Right-click the repository name in the Navigator and select **Close Client**.
2. Click **Yes** to confirm.

Note: If there are no open projects or jobs, the repository closes automatically without confirmation.

Adding and Editing a Repository Connection

Use the Repository Logon window to add a new repository connection or edit an existing connection.

1. Select **File > Open Repository**.
2. To add a new repository connection, click **Add**, enter the parameters for your new repository connection, and click **Save.**

   Note: To access the new repository, you must create at least one client and one client user definition.

3. To modify an existing repository connection, select the repository connection and click **Edit**, make changes, and click **Save**.

Creating a Client

Use the Repository Logon Window or User Accounts window to create a client.

1. In the Repository Logon window, enter a client name.
2. Enter a client user name. The name must be alphanumeric, can contain up to 255 characters, and cannot start with a number.
3. Enter a password.
4. Select **Register new user**.
5. If the client user is entitled to see all existing projects within the client, select **Show all objects**.
6. Click **Logon**.
7. Reenter the password and click **OK.**

See also

- *Creating a Client User* on page 15

Removing a Repository Connection

Remove a repository connection from the list in the Repository Logon window.

1. Select the repository from the Connection list and click **Remove**.
2. Click **Yes** to confirm the removal.

Setting Up and Maintaining User Accounts

Only registered client users can access a repository. You can register a client user in the Repository Logon window or in the User Accounts window.

Creating a Client User

Use the Users Account Window to create a user.

1. In the Navigator, right-click a repository name and select **User Accounts**.
2. Click **Add User**.
3. Enter the user name. The user name must:
    - Contain only alphanumeric characters
    - Start with an alphabetic character
    - Contain up to 255 characters
    - Not be empty

4. Enter a password.
5. Reenter the password.
6. Select **Show all objects** to show objects belonging to other repository users.
7. Click **OK.**

Removing a Client User

Use the Users Account Window to delete a client user.

1. In the Navigator, right-click a repository name and select **User Accounts**.
2. Select the user you want to remove and click **Remove User**.

   If the user is connected to the repository, click **Yes** to confirm that you want to remove the user and close all open projects and jobs.
3. Enter the password of the user you are removing and click **OK.**

Changing Passwords

Use the Users Account Window to change passwords.

1. In the Navigator, right-click a repository name, and select **User Accounts**.
2. Select the user for whom you are changing the password.
3. Click **Change Password**.
4. Enter the existing password of the user and the new password. Reenter the new password.
5. Click **OK.**

Transferring Objects Between Repositories

You can copy objects from one repository to another, maintaining references to the originating object.

Note: To transfer repository objects from the source repository to target repositories, all repositories involved must be accessible from the same Sybase IQ InfoPrimer Development instance.

Transferring Projects and Jobs

You can copy complete projects or jobs from one repository to another, maintaining references to the originating project.

1. In the Navigator, right-click the repository name or the project or job to transfer and select **Transfer**. Alternatively, select **File** > **Transfer**.
2. Select the source and target repository connections from the **Source session** and **Target session** list, respectively.

If the Transfer window is launched by selecting a repository, the corresponding repository connection is preselected as the source repository. If launched by selecting a project or job, the selected object, along with its children, appears selected for transfer.

To add a new source or target repository connection, click the **Open a New Session** icon.

Note: The information that appears in the Transfer Repository Objects window is refreshed each time a different source repository is selected. All connections opened from the Transfer Repository Objects window close automatically when the window is closed.

3. Select one or more objects to transfer. To display or refresh target repository information, click **Display or Refresh Target Access Information**.

   Selecting or unselecting a top-level folder automatically selects or unselects all folders under it. Selecting a job automatically selects all its projects. Objects that are selected more than once, for example projects used in jobs, are transferred only once. When a project is used in one or more jobs, selecting or unselecting the project in one location automatically selects or unselects the project in all other locations.

   You can search for source repository objects by specifying a search criteria based on a project or job name, source modifications timestamp to find all objects modified after the specified date, and the client user who last modified the source objects. Click the **Add Search Result to Transfer List** or **Remove Search Results From Transfer List** icons to select or unselect highlighted objects.

   Click **Transfer**.

Note: The transfer option copies project and job definitions. Related data, such as, parameter sets, execution properties, or database connection profiles are not transferred.

Alert Configuration for Runtime Events

Set alerts on the engine you are connected to using the Alert and Event Configuration window.

Select **Tools > Alert Manager** from the IQ InfoPrimer Development UI to bring up the Alert and Event Configuration window, which includes:

- Alert Services pane – select the engine to configure the alert on. By default, the selected engine is the one started by IQ InfoPrimer Development.
- Events pane – includes a navigation tree that displays different event types.
- Alerts pane – displays event to alert mappings.
- Alert Definition pane – used for configuring alerts.

Creating an Alert for an Event

Set up a new alert for a runtime event in the Alert and Event Configuration window.

Note: You can create multiple alerts for any event type for project and job execution. However, the alert name must be unique.

1. Select the engine on which to configure the alert.
2. Select the event type.
3. In the Alerts pane, click the **Add** icon.
4. Provide a unique name for the new alert and click **OK.**
5. Specify e-mail preferences for the new alert.
   a) Enter appropriate values in the To, CC, and BCC fields.
   b) You can manually enter the content in the subject or the body field, or click the **Select Event Property** icon. For example, if you select the [Job Name] event property, when the alert is triggered, the event property is replaced with "Job 1," which is your real job name.
   c) Click **Test Mail**.

   Note: The content of the subject and body in the test mail is generated automatically by the system.
6. Specify filter conditions for the new alert.
   a) Click the **Select Project or Job** icon and choose the project or job that should invoke the alert. Click **OK**.

      The filter condition is generated automatically. However, you can edit the filter condition.

      If you are entering the filter condition manually, click the **Select Event Properties** icon.
   b) Select **Include** to be notified when the events for the project or job match the filter condition. Select **Exclude** to be notified when the filter conditions are not met.
   c) Select **Enable alert during simulation** to be notified of an event during simulation. This option is available only for project events.
7. Click **Save.**

Editing an Alert

Modify an existing alert for an event in the Alert and Event Configuration window.

1. In the Alerts pane, click the alert.
2. Modify the preferences and click **Save**.

Deleting an Alert

Remove an existing alert using the Alert and Event Configuration window.
Select the alert to remove and click the **Delete** icon.

Copying an Alert (Save As)

Create a copy of an existing alert using the Alert and Event Configuration window.

1. Select the alert to copy.
2. Click the **Save As** icon.
3. Enter a name for the new alert.
4. Modify the e-mail preferences and filter conditions for the alert, if required. Click **Save.**

Note: Sybase recommends that you set up e-mail alert notifications on one machine. You can then manually copy the alert configuration file across different grid engines running on the network. Use a text editor to define the alerting configuration in the "SMTP" section of the `Default.ini` file in the `etc` directory.

Configuring the Alert Service on UNIX

Define e-mail alert notifications in IQ InfoPrimer. If you are configuring the alert service on a UNIX installation of IQ InfoPrimer Server, edit the `GridNode.ini` file.

Note: The alert service configuration is also required to use the **uSMTP** JavaScript function.

1. Navigate to the `etc` folder in the installation directory.
2. Use a text editor to open the `Default.ini` file and define the alerting configuration in the "SMTP" section, to match your SMTP server and account information:

   ```
   [SMTP]
   Server = SMTP Server name
   Port = port number of the SMTP Server
   Sender = account@SMTP Server
   Credential = user/password
   RetryCount = retry count
   RetryInterval = retry interval
   ```

   For example:

   ```
   Server = localhost
   Port = 25
   Sender = admin@localhost
   Credential = admin/7357506276773D3D
   RetryCount = 0
   RetryInterval = 5
   ```
3. Save and close the file.

Managing Database Connection Profiles

Database connection profiles allow specifying frequently used database connection details and storing them together with a profile name and a description in the repository for later use in EL and SQL Transformation projects.

Creating connection profiles supports faster and easier project design by applying the settings from a stored profile to respective project properties instead of entering them manually.

Projects can even hold a reference to stored profiles. Changing profile information automatically affects all projects referencing it when opened for design or execution, thereby supporting lifecycle management in these ways:

- Testing against different databases can be accomplished by changing a profile for all projects instead of creating parameter sets for each project.
- Switching from a development to a production environment can be accomplished by transferring the projects, given that the same profile with different settings exists on both repositories.

1. From the Sybase IQ InfoPrimer Development interface, select **Tools > Connection Profile Manager.**

   The Connection Profile Manager shows a list of all defined connection profiles for a selected repository session and a Profile Details pane, which is populated with data when you select a connection profile and where a profile setting is edited.
2. Create a new connection profile.
   - Select **Profile > New** or click the **Add Connection Profile** icon.
   - Enter a name, description, and the other connection details.
   - Click **Test Logon**.
   - Click **Save Connection Profile** to create the connection profile.

   The connection profile displays in the list.
3. To make changes to a profile, select it and edit it in the Profile Details pane. Click **Save** to save any changes or **Revert** to restore the original settings. To save it as a new profile select **Save As.**
4. To delete a profile, select the profile in the list, and click **Delete**.

Working with Profiles on Multiple Repositories

The Connection Profile Manager supports maintaining profiles in multiple repositories and exchanging profiles between repositories. When opened, the Connection Profile Manager shows a list of open repository sessions.

1. Use the **Choose session** drop-down list to switch between open repository sessions.

2. To add a repository connection to the list, click the **Open a new session** icon. These sessions close automatically when the window is closed.

   Switching the repository when a stored profile is selected has these effects:

   1. The profile list is updated to display the profiles stored in the selected repository.
   2. The profile list is searched for the same profile (transferred) that was selected previously.
   3. If the profile is found, it is selected causing the settings of this profile to be loaded into the Profile Details pane.
   4. If the profile does not exist in the current repository, the fields in the Profile Details pane continue to display the values.
3. Copy the profile to another repository.
   a) Select the profile to copy from the profile list.
   b) Switch to the target repository.
   c) Modify the profile details, if necessary.
   d) Click **Save As** to save the new profile .
4. Transfer the profile to the target repository.
   a) Select the profile to transfer from the profile list.
   b) Switch to the target repository.
   c) Modify the profile details, if necessary.
   d) Click **Save**.

See also

- *Transferring Objects Between Repositories* on page 16

Managing Grid Engines

Start, stop, register, and monitor grid engines.

Starting Sybase IQ InfoPrimer Server

Start the Sybase IQ InfoPrimer Server using the command prompt.

Change to the Sybase IQ InfoPrimer Server installation directory:

On Windows, enter:

```
GridNode
```

```
GridNode --port 5500
```

On Linux and UNIX, enter:

```
GridNode.sh
```

```
GridNode.sh --port 5500
```

The `GridNode.sh` script is generated by script `$SYBASE/SybaseIQIPServer/configure.sh`, which may be run during or after the IQIP Server installation.

On Linux and Unix, to update the Sybase IQ InfoPrimer Server configuration with:

- Sybase environment information, change to the Sybase IQ InfoPrimer Server installation directory and run the `configure.sh` file in the same session where the $SYBASE variables are set.
- Oracle environment information, change to the Sybase IQ InfoPrimer Server installation directory and run the `configure.sh` file in the same session where the $ORACLE_HOME and $TNS_ADMIN variables are set.
- IBM DB2 environment information, change to the Sybase IQ InfoPrimer Server installation directory and run the `configure.sh` file in the same session where the $DB2INSTANCE and $DB2DIR variables are set.

Starting Sybase IQ InfoPrimer Server as a Windows System Service

You can install and run Sybase IQ InfoPrimer Server as a Windows system service. To run it as a system service independently of the Windows GUI, start the IQ InfoPrimer Server after system start-up using a system user account.

Note: You must have administrator privileges to install, remove, start, and stop a system service.

To install the server as a Windows system service, enter:

```
GridNode.exe --install [additional parameters]
```

To remove the service, enter:

```
GridNode.exe --remove
```

When you run Sybase IQ InfoPrimer Server as a Windows service, basic events (failures, success messages, and so on) are written to the Windows event log.

Note: On Windows 7, you need to be an administrator to install the GridNode as a system service.

Stopping Sybase IQ InfoPrimer Server

You can stop a server from the console if it is a local or remote process. All currently running projects complete execution before the server stops.
On Windows, enter:

```
GridNode --shutdown
```

```
GridNode --shutdown --server [remotehost] --port [port]
```

On Linux and UNIX, enter:

```
GridNode.sh --shutdown
```

```
GridNode.sh --shutdown --server[remotehost] --port [port]
```

Note: To stop a grid engine running on a specified server and port, you must provide the server name and port number. If you do not provide the server name and port number, the local grid engine at the default port is stopped.

Grid Engine Registration

The grid architecture reduces job execution time by using parallel execution of projects on multiple distributed engines. After you install grid engines, you can register them for a special repository.

To leverage this scalability, you must:

- Install multiple grid engines
- Register your grid engines
- Prepare jobs for multiengine execution

Note: The terms “grid engine” and “IQ InfoPrimer server” are used interchangeably in this guide.

To register grid engines, select **Tools > Engine Manager**. If connections to multiple repositories are open, select one of them. The Engine Manager window displays a list of engines that are already registered for the selected repository.

The properties of a registered engine are:

- Name – user-defined name for the engine.
- Host – name or IP address of the engine host.
- Port – number of the port the engine is listening on.
- Base Rank – user-defined ranking for the engines. A job first tries to executes the projects on the highest ranked engines.
- Description – description for the server.

See also

- *Reducing Job Execution Time Using Multiple Engines* on page 209

Manually Registering a Grid Engine

You can manually register individual or multiple Sybase IQ InfoPrimer servers.

1. Select **Engine** > **New**, or click the **New Insert** icon.
2. Enter values.
3. Click **OK**.

Registering Multiple Engines

Register multiple engines using the New Engines window.

1. Select **Engine > New Network Search**.

   Note: The New Engines window displays all the engines running in the same subnet as Sybase IQ InfoPrimer Development. Grid engines not running in the same subnet as Sybase IQ InfoPrimer Development must be registered manually.
2. Select the engine to register.
3. Click **Add**.

Note: Click the **Refresh** icon on the toolbar or press the **F5** key to update and reload the list of grid engines.

Modifying an Engine Registration

Edit the engine registration in the New Engines window.

1. Select the engine in the list and choose **Engine > Edit**, or click the **Edit Selected Engine** icon.
2. Enter new values.
3. Click **OK.**

Deleting an Engine Registration

Remove an engine registration using the New Engines window.

1. Select the engine to delete.
2. Select **Engine > Delete**, or click the Delete selected engine icon on the toolbar.

Grid Engine Monitoring

The Engine Monitor displays information about all available grid engines in the environment, whether or not they are running or registered in the Engine Manager.

Note: You can use only engines that are registered in the Engine Manager for multiengine job execution.

To view information about the available grid engines running in the same subnet as IQ InfoPrimer Development, select **Tools > Engine Monitor**. On the Engine Monitor, use the **Update interval** field to specify the number of seconds to wait between two updates. The default value is 5 seconds.

Troubleshooting Grid Engine Issues

If you encounter issues with the grid engine, you can contact Sybase Technical Support for assistance.

Before you contact Technical Support:

1. Review the error text.
2. Review the log file.
3. Run the grid engine again with system trace enabled.
4. Verify the version and revision number as well as your machine ID using:

   ```
   GridNode --version
   ```

   Output:

   ```
   GridNode 15.3.0.32391
   ```
5. Verify the available licenses using:

   ```
   GridNode --licenses
   ```

   Output:

   ```

   GridNode 15.3.0.32391
   GridNode
   ----------------------------------------
   Product ID: InfoPrimerDevelopment
   Machine ID: 9TuZ+SoF62h4GC0JNQ==
   ----------------------------------------

   File       : sy_ip_development.license
   Product    : Sybase IQ InfoPrimer Development
   (InfoPrimerDevelopment)
   Version    : 15.3
   License    : Sybase InfoPrimer Features 15.3
   (SY_INFOPRIMER_DEVELOPMENT)
   Status     : Valid
   ```

Viewing Log Files

Use the File Log Inspector to view information about project and jobs execution, fatal errors, and to review the system log. Log files are located in the `\log` subdirectory of the installation directory.

1. Select **Tools** > **File Log Inspector**.
2. Click the log file information you want to view. You may see one or all of these log files:

- `alert.log` – captures history of the triggered alerts. Provide alert details, such as alert name, event type, date and time, and alert message.
- `execution.log` – captures information about job and project execution.
- `system.log` – captures information about system activities, and operational and exceptional events. You can check the error codes in the `system.log` file to determine the reason and possible solution for errors encountered while working with Sybase IQ InfoPrimer. The error codes that you may see in the `system.log` file include:

Error Code	Type	Description
0	Information	Job or project execution is successful
100 or 110	Error	IQ InfoPrimer engine initialization error
101	Error	Invalid license error
1100	Error	IQ InfoPrimer exception failure, including incorrect command line usage
1103 or 1104	Error	Failure due to an unspecified exception
10001	Error	Failure to retrieve information from repositories, including jobs, projects, and parameter sets
10005	Error	Job execution failed
10006	Error	Project execution failed
10007	Error	Project execution ending in a rollback state
10101	Error	Connection to the repository database failed

The detail level of data written to `system.log` depends on the trace level that has been set.

To set the trace level:

a) Select **Tools** > **Enable System Trace**.

b) Use the **uTracelevel**(*n*) function in a JavaScript procedure.

uTracelevel(*n*), where *n* is a value of 0 through 5, lets you set the trace level from within a component.

c) Specify the trace level in the `Default.ini` file in the `etc` subdirectory of the installation folder by modifying this line:

```
Tracelevel=value
```

where *value* is the trace level you want to set. You must restart Sybase IQ InfoPrimer for the change to take effect.

Note: Sybase recommends that you set a trace level to either 0 or 5.

3. (Optional) Click the **Truncate log** icon on the toolbar to truncate the log. Click **Yes** to confirm.
4. (Optional) Click the **Select rows containing a string or regular expression** icon on the toolbar to locate a particular log file.

Note: The Log File Inspector displays only the last 1MB of a log file. Use a text editor to view older records in a large log file.

Managing Projects

Basic project operations include creating, deleting, renaming, and saving. More complex operations include simulation (only for ETL projects) and execution.

Creating a Project

Create a new project using the Navigator or the File menu.
In the Navigator, right-click the *Project Type* folder and select **New >** ***Project Type***.
Alternatively, select **File > New >** ***Project Type***.

Modifying a Project

Edit an existing project using the Navigator.

1. In the Navigator, double-click the project to modify.
2. Make the changes and save the project.

Unlocking a Project

A project locked by another user client opens in read-only mode.
To make the project available for reading or writing, click **Unlock and Open** on the window that appears when you open a locked project. To unlock a project without opening it, right-click it in the Navigator and click **Unlock.**

Copying a Project

Create a copy or save an existing project using the Navigator or the File menu.

1. Double-click the project to copy to open it in the Design window.
2. In the Navigator, right-click the project and select **Save As**. Alternatively, select **File > Save As**.

   If you are working with multiple repositories, select the target repository.
3. Enter a name for the new project. A copy of an existing project is created, leaving the original untouched, and storing no references to the originating project.

Deleting a Project

Remove a project using the Navigator.

1. In the Navigator, right-click the project and select **Delete.**
2. Click **Yes** to confirm the deletion.

Renaming a Project

Rename a project using the Navigator.

1. In the Navigator, right-click the project and select **Rename**.
2. Enter a new name for the project and click **OK.**

Resetting Execution Properties

Reset execution properties to their default values using the Navigator.

1. In the Navigator, right-click the project, and select **Reset Execution Properties**.
2. Click **Yes** to confirm.

Managing Database Connection Profiles in EL and SQL Transformation Projects

The database connection profile created using the Connection Profile Manager can be used in EL and SQL Transformation projects. The settings from a profile can be applied to related project properties. You can also save settings from the database configuration windows of the EL and SQL Transformation projects as a new profile.

The database configuration windows of EL and SQL-T projects include a **Profile** list with stored connection profiles. To subscribe to a profile select it from the list. Alternatively, open the Connection Profile Manager, which when opened from the database configuration windows of EL and SQL-T projects shows an additional **Subscribe** button providing an alternate and advanced way of selecting a profile.

The **Subscribe** button is enabled only if the profile details displayed in the Profile Details pane is not modified, stored in the same repository as the project, and the interface is supported in the project.

When a project has a subscription to a profile the **Profile** list shows the respective entry selected. The connection setting project properties are populated with the current content from the stored profile and appear disabled.

Select **None** from the **Profile** list to unsubscribes and make the fields editable.

Database connection settings used in a project can be saved as a profile by opening the Connection Profile Manager. The Profile Details pane is populated with the current project property values and can be saved after a name and an optional description have been provided.

Note: When opening a project, which is subscribed to a non-existing profile, which is either deleted or not yet transferred, an initialization error message appears, and the project is unsubscribed.

Extract and Load (EL) Projects

The Extract and Load (EL) project is a multipurpose project for designing high-speed bulk extraction and bulk-loading processes.

The EL Project allows you to extract data from any of the supported datasources and load it into a Sybase IQ database.

An EL project facilitates loading multiple tables from a single database source or from multiple files. Sybase recommends that you use the EL project for migrating data from Oracle, Adaptive Server Enterprise, Microsoft SQL Server, flat files, and other supported datasources to a Sybase IQ database.

The EL project supports loading data into Sybase IQ from any of these supported file types:

- Sybase IQ compatible binary format
- Text files with delimited record values (including Apache Web log formats)
- Files with fixed length values and variable length records
- Large objects (LOBs)

Note: Files with fixed-length values and fixed-length records are not supported. For this particular file type, use ETL project instead.

Extracting LOB data is only supported for enhanced datasources namely, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, and LOB files (binary files or character files).

Configuring an EL Project

To configure an EL project define a datasource and the data destination, then select the tables to be extracted and loaded into Sybase IQ.

Note: The configured project can be used within a job that would use the EL project to load data into Sybase IQ, and one or more SQL Transformation projects to transform the data and load them into the final destination tables.

From the Navigator, right-click **Extract and Load Projects**, and select **New > Extract and Load Project** to start configuring an EL project.

Defining a Datasource

Define the datasource in the **Source** tab.

1. Select a type of datasource.

   Select **Database** or **File** as the datasource type.

2. If you select **Database** (default):

   - Select **Materialization with Replication Server** to use Replication Server® materialization methods for loading data. Using Sybase IQIP with Replication Server materialization allows you to coordinate initial data loading into Sybase IQ with auto-correction of source transactions during migration, and transform replicated data in Sybase IQ.

     See *Replication Server Administration Guide Volume 1 > Manage Subscriptions for details on materialization.*

     Note: To use Sybase IQ InfoPrimer with Replication Server for transformations, you must have Replication Server 15.6 ESD#1 or later.
   - Enter these connection details:
     - **Connection Profile** – to apply or subscribe to a previously defined database connection profile, select it from this list.
     - **Interface** – select the method or driver to use to connect to the source database.

       The **Interface** list includes an Insert Location option, which requires a valid destination database connection to be configured. When the Insert Location interface is selected, it displays a list of remote servers defined in the target IQ database (if the destination database has been defined), in the **Host** list. If there are no remote servers defined in the destination IQ database, configure a remote server and the necessary external logins on the IQ server.
     - **Host** – select the source database. The options that appear in this list depend on the selected interface.

       Note: If Oracle is the selected interface, the **Host** list may not display the Oracle source database if the ORACLE_HOME environment variable is not set correctly.
     - **User** – specify the authorized database user name.
     - **Password** – specify the authorized database user password. The password value is encrypted by default.
     - (Optional) **Database** – specify the database to be used as the source database. This option is not applicable for Oracle.
     - (Optional) **Schema** – specify schema or owner of the source tables.
     - (Optional) Click **Profiles** to save the entered connection details as a connection profile. Enter a profile name and description before saving the profile in the repository. Once the profile is saved, you can subscribe to it. See Database Connection Profiles in EL and SQL Transformation Projects.
     - The connection is automatically tested when you switch to another tab. If you want to manually test the connection, click **Test Logon**.

   Note: If you have previously created an EL Project, the database source connection details defaults to the most recently used source connection details.
3. If you select **Files**, enter:

- **Files** – add one or more source files to the EL project.
  - Click **Add File** and enter one or more comma-separated file paths. You can also browse for the files you want to add.

    Note: Empty file paths are not allowed.

    Click the **Edit** icon next to a selected file to open the Edit File dialog. You can edit the file path or browse to a different file. Multiple file paths and empty file paths are not allowed in the Edit File dialog.

    The Source File Path field supports Square Bracket Notation (SBN) expressions.
  - Click **Add Pattern** to create a file name pattern to match the file names in a specific directory. To add a file pattern, select a directory and enter a GLOB expression (patterns with * and ? wildcards) to match the source file names.

    You can enter Square Bracket Notation expressions in the Directory and Pattern fields. The value in the Directory field must evaluate to a directory path. Directory paths containing * or ? characters are not supported.

  Note: You can enter a Unix or Windows file path.
- **File format** – define a format for all the files and the files that match patterns, using the File Format editor. To open the File Format editor, click the **Define the File Format** icon.

  All files in an EL project must conform to a single file format. Separate EL projects are required if sources with different file formats need to be loaded.

  The File Format Summary pane shows a summary of the format specified in the format editor. For example, Byte Order appears if you selected IQ Binary as the file format. If there is an error in the file format definition, the File Format Summary pane displays the error instead.
- **Archive path** – specify where to archive the data files after the data has been successfully loaded into Sybase IQ. This can be a directory path or a path to a `zip` file. If you do not want to archive the source files, leave this field empty.

  Note: The Archive path field supports variables that are replaced with their actual values during runtime. These runtime variables are not SBN expressions and are not evaluated, and they must appear inside curly brackets when used.

  Select **Delete source files** to remove source files after successfully loading the source data into Sybase IQ (after archiving, if an archive path is set).

4. View the **Summary** pane for information about the source connection and error conditions, if any.

See also

- *Managing Database Connection Profiles in EL and SQL Transformation Projects* on page 30
- *Runtime Variables* on page 48
- *Configuring a File Format* on page 44

Defining a Data Destination

Define the data destination in the **Destination** tab.

1. Enter the destination database connection details. The destination database must always be an IQ server.

   Note: If you have previously created an EL Project, the database destination connection details defaults to the most recently used destination connection details.

   - **Connection Profile** – to apply or subscribe to a previously defined database connection profile, select it from this list.
   - **Interface** – select the method or driver to use to connect to the destination database.
   - **Host** – select the destination database. The options that appear in the this list depend on the selected interface.
   - **User** – specify the authorized database user name.
   - **Password** – specify the authorized database user password. The password value is encrypted by default.
   - (Optional) **Database** – specify the database to be used as the destination database.
   - (Optional) **Schema** – specify schema or owner of the destination tables.
   - (Optional) Click **Profiles** to save the entered connection details as a connection profile. Enter a profile name and description before saving the profile in the repository. Once the profile is saved, you can subscribe to it.
   - The connection is automatically tested when you switch to another tab. If you want to manually test the connection, click **Test Logon**.
2. View the **Summary** pane for information about the destination connection.

See also

• *Managing Database Connection Profiles in EL and SQL Transformation Projects* on page 30

Reviewing the Source and Destination Table Details

Verify the source and destination table details in the Tables tab.

Note: The information that appears on the Tables tab for file sources is different from the information that appears when using a database source.

The Tables tab shows all the tables or files from the datasource along with the destination table names to which each source is mapped. By default, the destination table names are the same as the source table or extracted from file names. You can enter or select a different destination table name.

If you have enabled Replication Server materialization in the Source tab, you cannot change the name of the destination table, which is the same as the source table with INSERT_RS as a suffix.

For the file datasource, all files that match a file pattern, by default, are loaded into the same destination table. To change this behavior, you can split the file pattern in the Tables tab.

Note: File patterns do not work for `HTTP` and `FTP`.

Perform these tasks on the database or file source type, and the destination tables:

- For database sources, click the **Add a Query Source** icon on the toolbar to add a query source to perform a transformation on the source data before it is loaded into Sybase IQ. Manually enter a SQL query in the Edit Query Source dialog, select lookup source tables and attributes, or open the Query Designer to draw the query and automatically generate the SQL statement. The default destination table name is the same as the name of the query source.

  To delete a query, right-click an existing source query entry in the `Transfer` table and select **Delete Query**.

  To edit a query, click the **Edit Query** icon next to the query in the Source column.
- Click the **Automatically Refresh when Source or Destination Definitions Change** icon to refresh the list automatically each time the source or destination connection details change. If the source type changes, the list is refreshed, regardless of the state of this icon.

  By default, this icon is set to "on" for all new project and "off" when opening an existing project.
- Click the **Include All Objects in Transfer** icon or **Exclude All Objects from Transfer** icon on the toolbar to select or unselect all tables for transfer. By default, all tables are selected for transfer. You can also select or unselect the **Transfer** option for each table individually.
- Click the **Mark All Transferring Objects for Truncation** icon or **Mark No Transferring Objects for Truncation** icon to select or unselect all destination tables for truncation before loading. By default, all selected destination tables are selected to be truncated before loading. You can also select or unselect the **Truncate** option for each table individually.

  Note: Destination tables can only be marked for truncation if the source table is selected for transfer.
- Click the **Create Missing Destination Tables** icon to create destination tables that do not yet exist on the destination database. See Create Missing Destination Tables.
- Click the **Truncate all Destination Tables** icon. Truncation and loading of destination tables occur in separate transactions.
- Click the **Drop all Destination Tables** icon. To truncate or drop the destination tables of a selected table, right-click the table row and select **Truncate Table** or **Drop Table**.
- Click the **Count the Records in all Sources** icon to count and retrieve the records in all of the selected source tables, views and queries and display it in the Source Count column. Alternatively, right-click the table name and select **Count Source**. This functionality is only available if the source type is a database source.

- Click the **Select Rows Containing a String or Regular Expression** icon to search for and select entries that match a GLOB or a Java regular expression. This is useful for selecting all tables that match a pattern and removing them from transfer.
- Specify a filter condition in the `Source Filter` field to filter the table entries based on the Source column values. Table entries that are not displayed, but were marked for transfer, continue to be loaded into Sybase IQ during execution. If you click **Transfer All**, **Transfer None**, **Truncate All**, and **Truncate None**, when a `Source Filter` is defined, only the settings of the visible rows are changed.

  Note: GLOB and regular expressions are not supported as filter conditions.
- Right-click a row and select **Load** to perform a test load of selected tables with a specified number of records.
- Right-click a row and select **Browse Source** to view source table or query data. This functionality is not available for file datasource.

  Note: If a destination table does not exist, an error icon appears in the Destination Table column of a table entry. To fix this, you can manually enter a destination table name of an existing table, select a destination table by clicking the **Table** icon in the respective column, or create the missing tables by clicking the **Create Missing Destination Tables** icon on the toolbar. If a connection to the destination database is not made, a question mark (?) appears in the Destination Table column of each entry.

Splitting a File Pattern

You can split a file pattern to load files or groups of files into separate tables.

1. In the File Pattern column of the transfer table, click the **Split Pattern** icon. You can also right-click any entry with a file pattern and select **Split Pattern**. The Split Pattern dialog appears.

   File patterns are split by entering a GLOB expression that matches all or a subset of the file pattern files. Each asterisk (*) and sequence of question marks (?) defines a part of the file name that can be used to group files together and define a destination table name. If there is more than one extracted part of the file name, you can select which parts to use. Parts are selected in the **Table Name Parts** table header if more than one part is available for selection.
2. Click **Add** to split the file pattern. The file pattern is split into multiple Split Pattern entries. The source in each entry may be a new file pattern or a file path, depending on the pattern split that you have defined. During execution, every file that matches the split pattern source (instead of the file pattern) is loaded into the selected destination table. Split pattern entries are determined during project design and not during project execution.

   Square Bracket Notation (SBN) expressions are not supported in the file name pattern of a split pattern entry. If a file pattern containing an SBN expression is split, the SBN expression is only evaluated when designing the split. The split pattern entries do not contain any SBN expressions. For example, if you split the file pattern, "[uGetEnv('MYDIR')]/[uGetEnv('MYPATTERN')]" with pattern "*.txt", the resulting split pattern entries is: [uGetEnv('MYDIR')]/<value of MYPATTERN>.txt.

Here are some examples for splitting file patterns.

Example 1: Loading into distinct tables

```
File Pattern: C:\data\*
Split Pattern: *.txt
Example file names: SALES.txt, ORDERS.txt, ...
Table Name Part: (only one available, automatically selected)
Destination Tables: SALES, ORDERS, ...
This loads each *.txt file in the C:\data\ directory into a
distinct destination table.
```

Example 2: Load into tables based on the table name part of a file name. This example creates split pattern entries sources such as, SALES_*.txt, ORDERS_*.txt, and so on.

```
File Pattern: C:\data\*.txt
Split Pattern: *_*.txt
Example file name: SALES_2010-01-01.txt, SALES_2010-02-01.txt,
ORDERS_2010-01-01.txt,
ORDERS_2010-02-01.txt
Table Name Part: First part selected
Destination Tables: SALES, ORDERS, ...
This loads all files with the same values in the selected parts
into the same destination table.
The selected parts also determine the default destination table
name.
```

3. To split a file pattern more than once, right-click a split pattern entry and select **Split Pattern**. When you split a pattern more than once, you actually split the file pattern, not the split pattern entries.
4. To remove a pattern split, right-click an entry and select **Remove Pattern Split**.
5. To modify a pattern split, click **Edit** in the File Pattern cell of an entry.

Creating Missing Destination Tables

Create all missing destination tables using the Table Creation window.

1. Click the **Create Missing Destination Tables** icon to create all destination tables that do not yet exist on the destination database. When the Table creation window displays, source metadata gets loaded for each missing destination table. After the metadata is loaded for a table, you can select the table and edit its attributes.

   If source metadata cannot be loaded for a destination table, the table is marked with an error icon and it does not have any attributes.

   If source metadata is not loaded for one or more tables in the destination table list, you can attempt to load metadata from tables in a database. Click **Get Metadata For All Tables** to load the metadata for all tables, or click **Get Metadata For Selected Tables** to load the metadata for selected tables in the list.

From the **Get Attributes From Database** window, select a Connection Profile or enter database connection information. Click **Test Logon** to the the connection. Click **Get Attributes** to load the metadata for each destination table. Metadata is loaded by matching the destination table name to the name of a table on the connected database. If a match is found, the matching table metadata is loaded for the destination table.

Note: If you have enabled Replication Server materialization in the Source tab, the **UPDATE**, **DELETE** work tables to be created also appear along with the list of destination tables.

2. Select one or more destination tables from the navigator.
3. In the Attributes pane, verify the attribute mapping of the selected table. Default mappings are marked with a question mark (?).
4. Specify a filter condition to filter the listed attributes.

   To filter the attributes, select the **Hide Mapped Attributes** option to hide all attributes that have been mapped to a destination datatype.

   You can filter on the source attribute name, description, and datatype. To apply one of these filters:

   - Select a filter from the **Add Filter** drop-down list.
   - Select '=' or '!=' from the **Operator** drop-down list.
   - Enter a GLOB expression in the **Filter** text field.
   - Click the **Add A Filter Condition To The Attributes List** icon to add the filter.

   You can also group unique datatypes by applying the Unique Data Type filter. All attributes that share the same datatype, length and scale are grouped together into a single entry in the **Attributes** table. While this filter is applied, any changes to the mapping of an attribute is also applied to attributes with the same datatype, length and scale.

   - Select Unique Data Types from the **Add Filter** drop-down list.
   - Click the **Add A Filter Condition To The Attributes List** icon to add the filter.

   Note: After applying the Unique Data Types filter, you cannot apply any more filters.

   The list of applied filters is displayed at the bottom of the **Attributes** table. Filtered columns display a small filter icon next to the column name. To remove all filters applied to the **Attributes** table, click **Clear All Filter Condition From The Attributes List** icon in the **Attributes** toolbar.
5. Map the attributes to a destination datatype by selecting one of these options:
   - Manual mapping – click and edit the target datatype, length and scale values in the **Attributes** table.
   - Map selected – select the attributes you want to map to the same data type, length, and scale in the **Attributes** table and click the **Map Selected Attributes To A Target Data Type** icon in the **Attributes** toolbar. Then, enter the target datatype, length and scale values and click **Map** to apply the mapping.
   - Map Visible – maps all attributes shown in the **Attributes** table to the same datatype, length, and scale. Click the **Map All Visible Attributes To A Target Data Type** icon

from the toolbar and enter the target datatype, length, and scale values. Then, click **Map** to apply the mapping.

- Map All – maps all attributes in all tables to the same data type, length, and scale. Click the **Map All Visible Attributes To A Target Data Type** icon in the **Attributes** toolbar and enter the target datatype, length, and scale values. Then, click **Map** to apply the mapping.

6. Click **Create**.

A progress bar displays the table creation progress as well as any errors that occur during the process. At any time, you can cancel the table creation process by clicking **Cancel**, which opens a dialog that gives you an option to drop the tables you have created. Click **Yes** to drop the new destination tables, or **No** to keep them.

Click **Close** to return to the Table Creation window. Any tables that could not be created are marked with an error icon. Any tables that were created are marked with a created icon and cannot be edited.

Click the **Reset To Default Target Data Types** icon to reset all table attribute mappings to their default mappings. This resets the mappings of tables that have not been created in the destination database.

Note: The Table Creation window does not attempt to create destination tables that do not have source metadata or do not have a complete attribute mapping.

7. Click **Close** to return to the Tables tab.

Note: If you attempt to execute an EL Project that is configured to load a non-existent destination table, the Table Creation dialog appears, prompting you to create the missing destination tables. After these tables are created, click **Close** to continue with the project execution, if the project is valid.

Specifying Processing Details

View a list of executables commands or scripts to be executed.

1. Enter the executable command or script along with their parameters. You can enter variables that are replaced with their actual values during runtime. These variables must appear inside curly brackets when used.

Note: Runtime variables are not SBN expressions and are not evaluated.

2. Specify the start time for execution.

- Load Start – at the start of the EL project execution.
- Table Start – at the start of table loading.
- Table Finish – at the end of table loading.
- Table Error – when an error occurs during table loading.
- Load Finish – when all tables are loaded and the execution of the EL project is about to finish.

- Load Error - at the end of loading in case of errors that occur during table loading.

  If the **Continue Load On Error** property is selected in the Advanced pane, execution continues even when a table error occurs.

See also

- *Runtime Variables* on page 48

Specifying Advanced Options

Specify how you want to execute the project in the Advanced tab.

Note: Options in the Advanced tab are set to appropriate default values. You must change these settings only for advanced use cases.

1. Select one of these execution modes:
   - Extract and Load
   - Extract Only
   - Load Only
2. Enter these details for the Extract and Load and Load only execution modes:
   - Select **Use client side load** to enable bulk loading of data into the IQ database from files located on client machines.
   - Select **Lock destination tables** to lock all destination tables before loading them in order to avoid deadlocks.
   - Select **Automatic load ordering** to determine the load order at runtime by examining the destination table dependencies.
   - In the `Archive path` field, specify the location where you want to archive the temporary data files after the data has been successfully extracted from the datasource.

     The Archive path field supports variables that are replaced with their actual values during runtime. These variables must appear inside curly brackets when used.

     Note: Runtime variables are not SBN expressions and are not evaluated.
   - Select **Archive load script** to save the IQ load script to a file in the archive location specified in the `Archive path` field.
   - In the `Derived name column` field, specify the name of the destination table column into which the source file or table name is to be loaded.
   - In the `Derived index column` field, specify the name of the destination table column into which the source record index is to be loaded.

     Note: The source record number may not match the order of a database source table.
   - In the `Derived load ID column` field, specify the name of the destination table column into which the execution Load ID is to be loaded.
3. Enter these details for all execution modes:

   Note: Sybase recommends to use the default values.

- In the `Extracted data directory` field, enter a temporary storage location for data extracted to files. Data is loaded into IQ from this temporary directory. If you do not specify a location (default), data is stored in the Sybase IQ InfoPrimer temporary directory on the execution machine.
- Specify the format of the temporary files used to store data extracted from a source (usually a source database), and before it is loaded into Sybase IQ from the **Stage file format** list. These files are created during the extract phase of the EL project execution.

  The loader internally stages the data into temporary files before sending the data to Sybase IQ using the **LOAD TABLE** command. You can initially populate a fresh and empty Sybase IQ database using these stage files.

  The IQ binary data format can be read faster by Sybase IQ, but takes longer to be written by Sybase IQ InfoPrimer, while the delimited text format is a bit slower when read by Sybase IQ, but faster for Sybase IQ InfoPrimer to generate. Sybase recommends using the delimited text format for better performance. However, when extraction and loading is done in two phases, select the IQ binary data format since loading into Sybase IQ is faster and the slower generation of the stage files do not affect the loading phase.
- Select the **Transaction type**, which can be one of:
  - Select **Single Transaction** to load each table as a single transaction. If selected, the transaction is rolled back in case of an error during the load of the table.
  - Select **Auto Commit** to directly write each block of data read from the source to target and then commit it.
  - Select **Periodic Commit**, which is similar to autocommit, to load data with a periodic commit interval, defined by the commit threshold.
  - Select **Restartable**, which is similar to periodic commit, to allow restarting of the project from a point of failure.
- Select **Commit threshold** to specify the number of records after which you want to commit data. This option is enabled only for the Periodic Commit and Restartable transaction types.
- Specify the number of records to be extracted at one time in the `Read block size` field. This property is not available for the Load Only execution mode.
- Select **Continue on error** to continue processing even if an error occurs when loading data into a database.
- Specify the number of records to be extracted and loaded from each of the source into the target database, in the `Row Limit` field. For example, if the source table has 10000 rows, and you just want the first 1000 rows to be loaded into the target database, you can enter 1000 in this field. The row limit is not used if set to zero (default).

  Note: This option is intended to be used for testing purposes only.

  - Select the **Pass-through optimizations** option to enable the tool to load data in the fastest possible way. Unselect this option if you want to prevent errors that may have been caused if you have selected this option and the tool has to make checks to see if the source is compatible with Sybase IQ. By default, the Pass-through optimizations option is selected and is disabled for EL Projects where the Source is set to IQ Binary formatted files.

- Click **Property options** to set the EL project properties.

See also

- *Parameter Sets* on page 209
- *Runtime Variables* on page 48
- *Enabling Client-Side Load Support* on page 127
- *Setting Project Properties* on page 47

Configuring Replication Server

Enter the configuration details for Replication Server in the Replication Server tab, if you have selected to load data using Replication Server materialization.

Note: This tab only appears if you have selected the Materialization with Replication Server option in the Source tab.

Enter the Replication Server connection details for the primary (source) database.

If the Replication Server for the replicate (target) database differs from the Replication Server for the primary (source) database:

- Unselect the **Same as Primary Replication Server** check box.
- Enter the Replication Server connection details for the replicate (target) database.

Configuring a File Format

Use the File Format Editor to configure file formats. The different types of configurable file formats are text files with delimited record values (includes Web Log format), files with fixed-length values and variable length records, IQ binary, and large objects.

Instead of configuring a format for every input file configure the format for all files.

1. Click the **Define the File Format** icon:
   - Sample Content pane – displays the contents of a sample document, for which you specified the source path in the Sample File Source field. You can detect delimited file formats by clicking the **Detect File Format** icon in the toolbar.
   - Format Properties pane – displays the file format properties and the metadata source.
   - Preview pane - displays a tabular view of data as it will appear in the destination table. Optionally, to read the column names from a data file, click the **Read Column Names from the Data File** icon in the Preview pane, provide the line number of the record that contains the column heading, and click **OK** to confirm. This option is only available to Delimited file formats.

Note: The File Format Editor displays a table preview only if the Metadata Source is valid for the selected sample file. For example, the destination table of a sample file (set in the **Tables** tab) must exist if the Metadata Source is set to **Destination Tables**.

2. Select the format type.

   - Select **Delimited** to read and extract structured data from source files containing delimited fields into a destination table in Sybase IQ. Configuring a delimited format in an EL project is similar to configuring a delimited file in the Text Data Provider component of the ETL project.
     - Specify the other mandatory and optional file properties for the delimited format type:
       - **Character encoding** – select the character set encoding with the correct endianness type for character data.
       - **Line delimiter** – specify how each row is delimited. Select the delimiter from the list or enter a different delimiter.
       - **Column delimiter** – specify how each column is delimited. Select the delimiter from the list or enter a different delimiter.
       - (Optional) `Null byte replacement` – set the character to replace null bytes.
       - (Optional) `Skip first rows` – specify the number of rows to be skipped in the row sequence.
       - (Optional) `Quote character` – specify the quote character pairs used in the file. You can define multiple pairs of quoting characters to support Web log formats.
       - Select **Ignore trailing empty columns** option if you want the trailing empty columns used in IQ parallel load files to be ignored.
       - Select the **Read empty values as null** option if you want all empty values to be loaded as null values in the destination table.
       - (Optional) Specify a **Metadata source**.
         - **Destination Tables** – uses the destination table to determine the table metadata.
         - **Database Tables** – uses a table on an external database to determine the table metadata. The external database table name must match the specified destination table name.
         - **Static Definition** – if metadata source is set to Static Definition, column names and datatypes default to "COL_#" and VARCHAR(255), respectively. If a column names row is specified, the column names from the source is used.

         You may require additional metadata to create destination tables, preview destination tables, or load file data into destination tables, depending on the format of the file. This additional metadata can be loaded from the IQ destination tables or the existing tables in another database server.

    - Delimited format metadata – requires no additional metadata to create destination tables or load delimited files into destination tables.
    - Fixed-length and IQ binary format metadata – since these formats are specific to each of the IQ destination tables, they require additional metadata to load files into the destination tables. The destination tables must already exist and you must not be required to define columns for each input file. To verify that fixed-length source files load correctly into their destination tables, you can preview the load result for each file in the File Format Editor.
    - Large objects (LOBs) format metadata - no additional metadata is required when loading LOBs into IQ or for creating the destination tables. The destination tables for LOB sources have the same table structure, created by Sybase IQ InfoPrimer.
- Select **Fixed Length** to read and extract structured data from source files containing fixed-length fields into a destination table in Sybase IQ.

  Note: The width of the fixed-length file columns must exactly match the column display width of the destination table.

  - Specify the other mandatory and optional file properties for the fixed-length format type:
    - **Character encoding** – select the character set encoding with the correct endianness type for character data.
    - **Line delimiter** – specify how you want each row to be delimited. Select the delimiter from the list or enter a different delimiter.
    - (Optional) `Null byte replacement` – set the character to replace null bytes.
    - (Optional) `Skip first rows` – skip a specified number of rows in the row sequence.
    - Select the **Read empty values as null** option if you want all empty values to be loaded as null values in the destination table.
    - For fixed-length files, the metadata source is always the destination tables. When specifying this format, the destination tables must exist.
- Select **IQ Binary** to extract and load structured data from an IQ binary file into a destination table in Sybase IQ.

  Specify the other mandatory and optional properties for the IQ Binary format type:
  - Byte Order – specify the byte ordering of the input files. Sybase IQ InfoPrimer assumes that the IQ Binary input files have the same byte ordering as the machine where the IQ server is running (default is DESTINATION). You can override the default by specifying HIGH (Big Endian) or LOW (Little Endian) depending on the machine architecture on which the files have been created.
  - For IQ Binary files, the Metadata Source is always the destination tables. When specifying this format, the destination tables must exist.

- Select **Large Object** to extract and load LOB from file into a destination table in Sybase IQ. EL Projects with LOB file sources, load file content into one or more LOB tables with a predefined structure containing CLOB or BLOB columns, depending on the LOB Type selection.
  - LOB Type – select BLOB (long binary) or CLOB (long varchar).
  - Sample Directory – specify a LOB directory and file name pattern in the Sample Content pane. A list of files in the directory appears. A preview of the resulting LOB table displays in the Preview pane.

See also

- *Text Data Provider* on page 78
- *Delimiter Considerations* on page 79

Setting Project Properties

Set evaluation, externalization, and encryption property options for the EL project.
Click **Property options** in the Advanced tab to set one of these properties:

- Property Evaluation – the Evaluate option lets you compute and evaluate dynamic property settings at runtime instead of assigning static values at design time. For some property items, the Evaluate option is selected, by default. To enable evaluation for a property, select the **Eval** check box next to the property. Once the Evaluate option is activated you can include JavaScript expressions in the corresponding field using square bracket notation (SBN) (where supported). The **Eval** check box is disabled for properties that do not support evaluation.
- Property Exernaliztion – the External option lets you expose properties as execution parameters, for use in Parameter Sets. For some property items, the External option is by default, selected. To externalize a property as an execution parameter, select the **Ext** check box next to the property.
- Property Encryption – EL Project data, as well as property values, are stored in the Sybase IQ InfoPrimer repository. Most of the records in the Sybase IQ InfoPrimer repository are not encrypted. For the password property, the Encrypt option by default, is selected. To encrypt property values, select the **Enc** check box next to the property.

See also

- *Evaluating SBN Expressions* on page 67

Runtime Variables

Use runtime variables to parametrize the executables commands or scripts in the Processing tab or to specify archive paths in the Source and Advanced tabs.

Note: These variables must appear inside curly brackets when used. Runtime variables are not SBN expressions and are not evaluated.

The list of supported runtime variables include:

- {JobName}
- {JobID}
- {ProjectName}
- {ProjectID}
- {ExecutionID}
- {Timestamp} – a timestamp values in a format that can be used for directory or file names.

  Example: If you want to archive each stage file grouped by time, use:

  ```
  c:\archive\{Timestamp}
  ```
- {Target} – name of the current target table.

  Example: If you want to archive each stage file grouped by job name and target table name, use:

  ```
  c:\archive\{JobName}\{Target}\{ExecutionId}
  ```

  Note: Missing directories are created at runtime.
- {Source} – name of the current source table.
- {Event} – name of the current event (BeginLoad, EndLoadSuccess, EndLoadError, BeginTable, EndTableSuccess, EndTableError). Only used in the Processing tab.
- {TargetOwner} – owner part of a qualified target table name.

  Example: USER for a table called USER.CUSTOMER
- {TargetTable} – table part of a qualified target table name.

  Example: CUSTOMER for a table called USER.CUSTOMER
- {SourceOwner} – owner part of a qualified source table name.

  Example: USER for a table called USER.CUSTOMER
- {SourceTable}– table part of a qualified source table name.

  Example: CUSTOMER for a table called USER.CUSTOMER
- {ScriptPath} – current script directory. Equivalent to the SBN expression [uSystemFolder('APP_SCRIPT')].

SQL Transformation Projects

A SQL Transformation project generates transformations to be performed on data that exists in a Sybase IQ database.

A SQL Transformation project is specialized to perform SQL transformations where the source and target are both on the same instance of Sybase IQ. As compared to JavaScript transformations, SQL transformations executed in Sybase IQ can enhance performance and scalability.

Setting up the IQ connection – A SQL Transformation project defines the Sybase IQ connection at the project level. The entire transformation project is translated into a single SQL script, and all transformation tasks use the same connection. If you have previously created a SQL Transformation project, the connection details defaults to the most recently used destination connection details. A SQL Transformation project can use database connection profiles that have been defined previously.

A SQL Transformation Project controls the flow of execution, based on the connections between components. You can add the SQL Transformation components from the Component Store to the Design window, connect them, and configure them by setting properties in their respective configuration windows.

See also

- *Managing Database Connection Profiles in EL and SQL Transformation Projects* on page 30

SQL Transformation Project Properties List

Connection parameters and other items to define for a SQL Transformation project.

Table 1. Required Properties

Property	Description
Interface	Specify the method or driver to use to connect to the Sybase IQ target.
Host Name	Specify the host where the Sybase IQ target is running.
User	Identify an authorized database user.

Optional Properties

The Optional Properties table identifies the properties that are not manadatory for a SQL Transformation project.

Property	Description
Profile	Subscribe to a database connection profile. See Managing Database Connection Profiles.
Password	Authorize database user and protect the database against unauthorized access.
Database	Identify the database to use as datasource.
Schema	Identify the schema or owner you want to use as datasource.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
SQL Script	Generated SQL script sent to database for execution.
SQL Stored Procedure	Store the automatically generated stored procedure.
Transactional	Set to perform all operations in a single database transaction that is committed when the project finishes normally.

See also

- *Managing Database Connection Profiles* on page 20

SQL Transformation Components

SQL Transformation components include Start, Task Group, Decision, Finish, and Error.

Start

The Start component marks the beginning of an execution flow.

The Start component has no input ports. It has a single output port that accepts only one connection to either a Task Group or Decision component.

Since each SQL Transformation project needs a single Start component, it is automatically added to each project.

The Start component allows you to create user-defined variables, for which you are required to provide a name, type, and an optional initial value. Double-click the **Start** component in the Design window or select the component and click the **Variables** icon in the Properties window, to access the Variable Editor. All variables are defined at the project level and can be used by other components in the same project.

Table 2. Start Component Compatibility with Other SQL Transformation Components

Component	Input	Output
Task Group	No	Yes
Decision	No	Yes
Finish	No	No
Error	No	No

Task Group

The Task Group component represents execution of one or more transformation tasks in the control flow.

Transformation tasks define data flows that are executed as part of a larger SQL Transformation project. Each transformation task is a logical transformation unit and can be translated into a complete SQL statement or script. A Task Group allows you to execute all tasks together as one transaction, or in individual transactions.

The transformation tasks that you can define within a Task Group component are:

- Generic SQL
- Custom SQL

For Task Groups and individual tasks, you can specify these properties:

- Critical task – terminate a transformation project immediately in case of a failure. The Critical Task property applies to individual tasks.
- Single transaction – execute all tasks in a Task Group together as one transaction , or in individual transactions . This option is applicable only when project level transaction is not selected. It applies to the Task Group

  When Single transaction is not selected, each task in the group is executed in its own transaction, thus individually committed or rolled back. If a task flagged as Critical fails the project is aborted. The Continue on error property does not have any effect.

  When Single transaction is selected, a failure of any task in the group rolls back the transaction and finishes execution of the group. In this case, the Critical task option has no effect.
- Continue on error – continues project execution even if an error occurs when executing non-critical tasks contained in the Task Group. This property applies to the Task Group.

A Task Group has a single input port that accepts multiple connections from Start, Task Group, and Decision components. It has a single output port supporting a single connection to a Task Group, Decision, Error, or Finish component.

Table 3. Task Group Component Compatibility with Other SQL Transformation Components

Component	Input	Output
Start	Yes	No
Finish	No	Yes
Error	No	Yes
Decision	Yes	Yes
Task Group	Yes	Yes

Custom SQL Transformation

A Custom SQL transformation task lets you write, edit, or import a SQL script or batches of scripts that perform a custom transformation.

Instead of modeling, you can use a SQL script to describe a transformation. You can use a Custom SQL task to model uncommon or extremely complex transformation types.

Configuring a Custom SQL Transformation Task

Define a custom transformation script using the Task Group component.

1. Add the Task Group component to the design window, right-click the component on the design window, and select **Add task** to create a new transformation task. Alternatively, double-click an existing Task Group component or with a Task Group component selected in the design window, click the **Tasks** icon in the Properties window, and click the **Create a new task** icon.
2. Select **Custom SQL** as the task type. Enter a name and description for the task and click **Create.**
3. Create the SQL script to describe a transformation. You can:
    - Manually enter the script in the SQL Script text area of the editor.
    - Drag and drop the references and functions you want to add to your script from the left pane to the text area.
        - The Model tab lists all the tables, views and stored procedures that you can use.
        - The Reference tab lists all available IQ functions and project variables that you can add to the script.
4. (Optional) Import or Export the SQL script from or to a file.
    - To import the SQL script from a file, click the **Import From File** icon, select the file to import, and click **Open.**
    - To export the script to a file, click the **Export to File** icon, provide a file name, and click **Save.**

5. Save the script and run it by clicking the **Execute a Script** icon. Alternatively, select **Execute** from the **Script** menu.

Generic SQL Transformation

A Generic SQL transformation task provides a user interface for modeling transformations and generating SQL scripts for the transformation.

Select the generic SQL transformation task, which is optimized to work with large number of tables and attributes, to design SQL transformations.

Configuring a Generic SQL Transformation Task

Define a generic SQL transformation script using the Task Group component.

1. Add the Task Group component to the design window, right-click the component on the design window, and select **Add task** to create a new transformation task. Alternatively, double-click an existing Task Group component or with a Task Group component selected in the design window, click the **Task** icon in the Properties window, and click the **Create a New Task** icon.
2. Select **Generic SQL** as the task type. Enter a name and description for the task and click **Create.**

   The Generic SQL Transformation Editor appears, which includes the Transformation Definition, Dictionary, and Transformation panes.

   - Dictionary – displays all the tables and views on the Model tab, and the recently used tables or views on the Recent tab. You can set the default number of tables or views that you want to view under the Recent tab, in the **File > Preferences** window under Query Designer.
   - Transformation Definition – includes the Design window, which is used to automatically generate **SQL** statements that are specific to the records you are working with. The Design window contains these tabbed panes to display different parts of the SQL transformation:
     - Tables – define source tables and joins.
     - Joins – defines join operations and the join ordering.
     - Attributes – defines attributes and their transformation.
     - Filters – defines the WHERE condition.
     - Sort – defines the SORT BY condition.
     - Group – defines the GROUP BY condition.
     - Target – determines the kind of SQL statement to be generated, like **INSERT**, **UPDATE**, **DELETE** operations, defines the destination table and the attribute mappings.

       Note: Source attributes are mapped to target attributes based on their order in the Attributes tab. You may need to change the order of attributes in the Attributes tab to match the order specified in the Target tab. This ordering also affects any target attributes used in key joins between the source query and the destination table.

- Transformation – set source query attribute usage, view the attribute details, as well as the generated transformation at any point of the script creation.

Decision

A Decision component is a rule-based component that branches the execution flow based on SQL expressions.

A Decision component contains one or more SQL-based rules that determine which branch of a control flow is executed. Each branch is mutually exclusive. Error handling branches are also created by defining appropriate split rules in a Decision component.

A Decision component has a single input port that accepts multiple connections from the Start, Task Group, and Decision components, and two or more output ports each supporting a single connection to a Task Group, Decision, Finish, or Error component.

You can add rules and define the evaluation order using the configuration window. Each rule (condition) is associated with an output port. They are evaluated in the order you specify. You can define a SQL query including functions and variables or call a stored procedure in a rule, as long as it returns a single value for evaluation. The value of a user-defined variable set as an increment variable automatically increments each time the associated branch is executed. By default, when you add a Decision component, a blank rule applies that represents an ELSE condition. If any of specified rules is not true, then the blank rule is executed.

For example, the rule on adding a Decision component by default includes two output branches:

- Branch 1 – @project_status = 'SUCCESS'
- Branch 2 – no rule exists, this is the ELSE condition (default)

Listed are some examples for the Decision component:

- Example 1

```
@projectstatus = ‘SUCCESS’
This expression is composed of a variable an operator and a
literal value.
The global variable @projectstatus is built-in and available for
the user in
decision expressions.
Valid values for @projectstatus include ‘SUCCESS’ and ‘ERROR’.
```

- Example 2

```
(SELECT COUNT (*) FROM Customers_Xfrm) >  0
This expression is composed of a select, a comparison operator and
a literal value.
Parentheses around the SELECT are required for IQ.
```

- Example 3

```
@projectstatus = ‘ERROR’ AND @retryCount = 0
This expression is composed of two variables, two literal values,
two comparison operators,
and a logical operator.  The second variable (@retryCount) is an
example of a user-defined variable.
```

- Example 4

```
CustXfrmHasData() > 0 AND @projectStatus = 'SUCCESS'
This expression integrates a user-defined function.
```

- Example 5

```
CustXfrmSuccess() = 0 AND CustXfrmHasData() > 0
This expression is composed of two user-defined functions.
```

Table 4. Decision Component Compatibility with Other SQL Transformation Components

Component	Input	Output
Start	Yes	No
Finish	No	Yes
Error	No	Yes
Task Group	Yes	Yes
Decision	Yes	Yes

To add the Decision component to a SQL Transformation project, drag it from the Component Store onto the Design window.

Finish

The Finish component visually represents the end of a successful execution of a SQL Transformation project.

The Finish component has no output ports. It has one input port that accepts multiple connections from Task Group and Decision components.

A SQL Transformation project can have multiple Finish components but Sybase recommends that you to include only one in each project. However, adding more than one Finish component does not affect the execution of the SQL script.

Table 5. Finish Component Compatibility with Other SQL Transformation Components

Component	Input	Output
Task Group	Yes	No
Decision	Yes	No
Error	No	No

Component	Input	Output
Start	No	No

To add the Finish component to a SQL Transformation project, drag it from the Component Store onto the Design window.

Error

The Error component visually represents the end of a failed execution of a SQL Transformation project. At runtime, an Error component terminates the execution.

The Error component has no output ports. It has one input port that accepts multiple connections from Task Group and Decision components.

A SQL Transformation project can have multiple Error components, but Sybase recommends that you include only one in each project.

Table 6. Error Component Compatibility with Other SQL Transformation Components

Component	Input	Output
Task Group	Yes	No
Decision	Yes	No
Finish	No	No
Start	No	No

To add the Error component to a SQL Transformation project, drag it from the Component Store onto the Design window.

Sample SQL Transformation Projects

Perform transformation operations, such as Delete, Insert, Update, and Upsert using the SQL Transformation project.

Performing Delete Operations Using a SQL Transformation Project

Create a sample SQL Transformation project to delete data out of a table in Sybase IQ.

1. From the Navigator, right-click **SQL Transformation Projects** and select **New > SQL Transformation Project**.
2. Specify the database connection parameters. If you have a previously defined connection profile for the database connection, select it from the **Profile** list. This list displays the

Sybase and ODBC profiles stored in the same repository as the project in the Database Configuration window and the Property window.

3. Drag the **Task Group** component onto the Design window and select **Generic SQL** as the task type. Enter a name and description for the task and click **Create**.
4. If necessary, click the output port of the Start component and drag it onto the input port of the Task Group component to create a connection. This connection is made automatically if the **Create links automatically when components are added** preference is selected.
5. Double-click the **Task Group** component to open the Task Container editor.
6. Select the task you created and click the **Edit the Selected Task** icon to open the Generic Transformation editor, where you can define the transformation rules. Before opening the Generic Transformation editor select one of these:
   - **Critical** – specify each task as critical or noncritical.
   - **Single transaction** – execute all tasks in a group as a single transaction.
   - **Continue on error** – continue processing even if an error occurs.
7. On the Target tab, select **DELETE** as the transformation operation.
8. Drag a destination table from the Model tab to the **Destination** text field. For example, DEMOUSER. SALESDETAILS.

   When you select the transformation operation and the destination table, a SQL statement is generated automatically and appears in the Transformation pane under the Generated Transformation tab.
9. Add a **WHERE** clause to this transformation.
   - Click the **Tables** tab under the Transformation Definition pane.
   - Select a table from the Model tab and drag it onto the Design window. For example, DEMOUSER. SALESDETAILS.
   - Click the *source_table_name* tab under the Transformation pane and select the attributes to add to the generated transformation. For example, under the DEMOUSER. SALESDETAILS tab, select the **WHERE** option to filter or add a **WHERE** clause on the Quantity attribute.
   - Go to the Filters tab under the Transformation Definition pane and enter a filter condition. For example, to delete records from the DEMOUSER. SALESDETAILS table with a quantity less than or equal to 500, enter <= as the Operator and 500 as the Value.
10. Click the **Generated Transformation** tab to view the regenerated SQL statement. In this case, the generated transformation is updated with the **WHERE** clause. Click **Save** to return to the project window.
11. Drag the **Decision** component onto the Design window. Create a link between the Task Group component and Decision component.
12. Drag the **Finish** component onto the Design window. Connect the input port of this component to the Success output port of the Decision component.

13. Drag the **Error** component onto the Design window. Connect the input port of this component to the Error output port of the Decision component.
14. Save the project. Click **Execute** on the toolbar to run the project.

    Click **Close** after the project successfully executes.

    You can view and edit the project generated SQL script by clicking the **SQL Script** icon in the Properties window. However, if you regenerate the script, you lose the changes you have added in.

    Note: SQL Transformation projects also support key based delete whereby target table data may be deleted if the attributes selected as keys in the Target table match the attributes mapped in the source query. To work properly, you must add attributes to the Attributes tab of the source query in proper order.

Performing Insert Operations Using a SQL Transformation Project

Create a sample SQL Transformation project that inserts data into a Sybase IQ table.

1. From the Navigator, right-click **SQL Transformation Projects** and select **New > SQL Transformation Project**.
2. Specify the database connection parameters. If you have a previously defined connection profile for the database connection, select it from the **Profile** list. This list displays the Sybase and ODBC profiles stored in the same repository as the project in the Database Configuration window and the Property window.
3. Drag the **Task Group** component onto the Design window and select **Generic SQL** as the task type. Enter a name and description for the task and click **Create**.
4. If necessary, click the output port of the Start component and drag it onto the input port of the Task Group component to create a connection. This connection is made automatically if the **Create links automatically when components are added** preference is selected.
5. Double-click the **Task Group** component to open the Task Container editor.
6. Select the task you created and click the **Edit the Selected Task** icon to open the Generic Transformation editor, where you can define the transformation rules. Before opening the Generic Transformation editor select one of these:
    - **Critical** – specify each task as critical or noncritical.
    - **Single transaction** – execute all tasks in a group as a single transaction.
    - **Continue on error** – continue processing even if an error occurs.
7. On the **Target** tab, select **INSERT** as the transformation operation.
8. Drag a destination table from the Model tab to the **Destination** text field. For example, DEMOUSER. SALESDETAILS.

    When you select the transformation operation and the destination table, a SQL statement is generated automatically and appear in the Transformation pane under the Generated Transformation tab.

9. Specify the source tables.

- Click the **Tables** tab in the Transformation Definition pane.
- Click the Model tab, then select and drag the three tables to subselect from onto the Design window. For example, DEMOUSER.STORES, DEMOUSER.SALES, and DEMOUSER. SALESDETAILS.

10. Specify a **JOIN** criteria for the added source tables based on attributes. For example, create a join between the DEMOUSER.STORES table and DEMOUSER.SALES table based on STORE_ID and between the DEMOUSER.SALES and DEMOUSER.SALESDETAILS based on the ORDER_NUM. You can specify multiple joins between attributes.

11. Click the *source_table_name* tab in the Transformation pane and select the attributes to add to the transformation. For example, click the **Select** option:

- For STORE_ID attribute under the DEMOUSER.STORES tab.
- For ORDER_NUM attribute under the DEMOUSER.SALES tab.
- For TATTLED, DISCOUNT, and QUANTITY attributes under the DEMOUSER.SALESDETAILS tab.

Note: Selected attributes in the Attributes tab must be listed in an order that matches the included attributes in the Target tab, to make sure the generated transformation is correct.

12. Go to the Attributes tab under the Transformation Definition pane and enter the functions to perform on an attribute. For example, to change the ORDER_NUM attribute to lowercase, enter **lcase** in the **Function** column.

13. Add a **WHERE** clause to this transformation. For example, click the **DEMOUSER.STORES** tab in the Transformation pane and select **Where** for the STORE_ID attribute. To add a filter condition, click the **Filters** tab and specify a filter criteria.

14. To sort the records based on a particular attribute, click the respective *source_table_name* tab and select the **Sort** option for that particular attribute.

15. Click the **Generated Transformation** tab to view the regenerated SQL statement. In this case, the generated transformation is updated with the **JOIN**, **WHERE**, **ORDER BY** clause and the attribute details. Click **Save** to return to the project window.

16. Drag the **Decision** component onto the Design window. Create a link between the Task Group component and Decision component.

17. Drag the **Finish** component onto the Design window. Connect the input port of this component to the Success output port of the Decision component.

18. Drag the **Error** component onto the Design window. Connect the input port of this component to the Error output port of the Decision component.

19. Save the project. Click **Execute** on the toolbar to run the project.

Click **Close** after the project successfully executes.

You can view and edit the project generated SQL script by clicking the **SQL Script** icon in the Properties window. However, if you edit and regenerate the script, you lose the changes added in.

Performing Update and Upsert Operations Using a SQL Transformation Project

Create a sample SQL Transformation project that performs update and upsert operations.

1. From the Navigator, right-click **SQL Transformation Projects** and select **New > SQL Transformation Project**.
2. Specify the database connection parameters. If you have a previously defined connection profile for the database connection, select it from the **Profile** list. This list displays the Sybase and ODBC profiles stored in the same repository as the project in the Database Configuration window and the Property window.
3. Drag the **Task Group** component onto the Design window and select **Generic SQL** as the task type. Enter a name and description for the task and click **Create**.
4. If necessary, click the output port of the Start component and drag it onto the input port of the Task Group component to create a connection. This connection is made automatically if the **Create links automatically when components are added** preference is selected.
5. Double-click the **Task Group** component to open the Task Container editor.
6. Select the task you created and click the **Edit the Selected Task** icon to open the Generic Transformation editor, where you can define the transformation rules. Before opening the Generic Transformation editor select one of these:
   - **Critical** – specify each task as critical or noncritical.
   - **Single transaction** – execute all tasks in a group as a single transaction.
   - **Continue on error** – continue processing even if an error occurs.
7. On the Target tab, select **UPDATE** as the transformation operation to perform an update operation and select **UPSERT** to perform an upsert operation.

   Note: Note: For improved performance in Sybase IQ, the UPSERT functionality is designed as a delete and an insert operation. To create a traditional UPSERT functionality, use the Custom SQL transformation task.
8. Select a destination table by dragging the table from the Model tab to the **Destination** text field.

   When you select the transformation operation and the destination table, a SQL statement is generated automatically and appear in the Transformation pane under the Generated Transformation tab.
9. Select the attributes of the target table to update. For an update operation, you may also enter an expression or an update value for each attribute.

Note: By default, source attributes are mapped to target attributes based on the order of attributes listed in the Attributes tab.

10. For key based updates, select the key columns. This is optional in case of an update operation but manadatory for an upsert operation. For upsert, the key is used as a join index to delete from the destination before data is inserted from the source.

Note: All attributes with the Key or Include options selected, are inserted.

11. Specify the source tables.
    - Click the **Tables** tab.
    - Click the **Model** tab, select, and drag a source table onto the Design window.

    Note: Source query attributes must be added to the Attributes tab in an order matching the included attributes in the Target tab, for proper mapping and key join generation.

12. In the Transformation pane select the *source_table_name* tab to view the table columns. For update, select the columns to use to match the key columns selected on the destination table. For upsert, select the columns to use for insertion and to match the key columns selected on the destination table.
13. (Optional) Select the **Where** option for columns on which you want to add a **WHERE** clause. To add a filter condition, click the **Filters** tab and specify a filter criteria.
14. Click the **Generated Transformation** tab to view the regenerated SQL statement. Click **Save** to go back to the project window.
15. Drag the **Decision** component onto the Design window. Create a link between the Task Group component and Decision component.
16. Drag the **Finish** component onto the Design window. Connect the input port of this component to the Success output port of the Decision component.
17. Drag the **Error** component onto the Design window. Connect the input port of this component to the Error output port of the Decision component.
18. Save the project. Once the project is saved successfully, click **Execute** on the toolbar to run the project.

    Click **Close.Close** after the project successfully executes.

    You can view and edit the project generated SQL script at any point of time by clicking the **SQL Script** icon in the Properties window. However, if you edit and regenerate the script, you lose the changes added in.

Troubleshooting a Failed SQL Transformation Project

Debug a failed SQL Transformation project.

- Use the log viewer to view the various grid node logs.

These logs files are found in the `logs` directory of the installation folder and can be viewed by selecting **Tools > File Log Inspector**.
- Use the Sybase IQ log files for the Sybase IQ servers issues.
- Use the Execution and Log Manager from the **Tools** menu to view details of individual executions of jobs or projects.
- Turn on tracing for the grid node and Sybase IQ servers.
- Execute the transformation task SQL script individually to check for errors. You can also execute the SQL script in another application like **dbisql** to check for errors.
- Contact Sybase Technical Support for assistance if you encounter issues with the grid engine.

See also

- *Viewing Log Files* on page 25
- *Troubleshooting Grid Engine Issues* on page 25

Extract Transform and Load Projects

An Extract Transform and Load (ETL) project extracts data from multiple heterogeneous datasources, applies a series of transformation rules or functions to the extracted data to derive the data for loading into the end target, and loads the data into one or more IQ data targets.

Project Customization

You can create data transformation projects without using programming code or SQL statements.

For example, you can:

- Use the Query Designer to generate **select** statements inside queries, lookup definitions, preprocessing, and postprocessing SQL.
- Use the data mapping features of the links between the components to map attributes between datasources and data sinks.
- Use the built-in **Create Table From Port** command of the component you are using, to create temporary or persistent staging tables, or to create tables in the destination database.
- Use the Content Explorer to browse schema information and data content of all connected datasources.
- Use the XML via SQL Data Provider component to read hierarchical XML documents and automatically generate a relational structure.
- Schedule execution of projects and create jobs within Sybase IQ InfoPrimer Development.

Additionally, when you have complex data transformation requirements, you can use:

- Manually optimized SQL **select** statements to adjust the data extraction process.
- SQL statements to apply data manipulation commands in different phases of the transformation.
- JavaScript to write procedures, perform complex calculations, or manipulate objects in the operating system environment.
- Indirection via expressions to dynamically control your projects by using environment or user variables.

JavaScript

Sybase IQ InfoPrimer supports the JavaScript language for expressions and procedures used in components, to transform and manipulate data within the transformation process.

You can also use JavaScript expressions enclosed in square brackets within component property values allowing a parameterized configuration. Square bracket notation (SBN) is a widely applicable indirection mechanism within the Sybase IQ InfoPrimer environment. You can apply square bracket notation, for example, within SQL statements and file name specifications to compute and assign values dynamically at runtime.

Datatypes and Data Formats

Datasource datatypes are preserved during transformation.

Internally, Sybase IQ InfoPrimer distinguishes between string and numeric datatypes. The Standardize data format option of the data providers or data sinks automatically converts data to a standard format, which is then converted to a format, which the target database can process. You need not manually convert various date and number formats when working with different databases.

By default, the Standardize data format option is selected. However, if you experience problems with date or number fields, you can disable this option and manually convert the data using the Preference window.

Unicode Support

All components process and support Unicode and multibyte data. You can use Unicode-enabled transformation functions in calculations, scripts, and procedures.

The level of Unicode support for Sybase IQ InfoPrimer allows you to:

- Extract, transform, and load data containing Unicode characters
- Use Unicode characters in component properties:
  - File or directory names
  - Metadata, such as, table or attribute names
  - Connection settings, such as, database, schema, user, or password
  - Transformation rules

SQL Statements

Most of the data delivered by Data Providers is defined by using SQL statements stored in the Query property. Sybase IQ InfoPrimer supports a modified set of the SQL92 standard.

You can manually write or copy SQL statements from existing projects into the **Query** property. You can use the Query Designer to draw the query and automatically generate the SQL statement.

SQL Queries and Commands Execution

You can enter and execute a SQL query or a series of SQL commands for databases associated with Source, Lookup, Staging, and Destination components.

Executing SQL Queries

Enter and execute a SQL query for databases associated with Source, Lookup, Staging, and Destination components.

1. In the Design window, right-click the component and select **Execute SQL Query**.
2. Enter a **select** statement in the Query field. You can use any valid SQL notation of the connected database to build the query. To create the query using tables and views from the available database schema, click the **Open the Database Lookup** icon.
3. Click the **Execute the Query** icon.

   On successful execution, the result appears in the Data Viewer window.

Executing SQL Commands

Enter and execute a series of SQL commands for databases associated with Source, Lookup, Staging, and Destination components

1. In the Design window, right-click the component and select **Execute SQL Commands**.
2. Enter the SQL commands in the Command field. To create the command using tables and views from the available database schema, click the **Open the Database Lookup** icon.
3. Click the **Execute the Commands** icon.

   Note: Executing a series of commands does not return a result set.

Project Execution

You can run an ETL project using either simulation or execution mode.

Both modes perform all functions of the components included in a project, including the physical transfer of data into the data targets, which are also called data sinks.

Execution mode lets you:

- Run ETL projects that have been saved to the repository. Unsaved changes are not executed.
- Execute the ETL project and reflect the changes in the data sink. You cannot monitor the transformation process step by step.

Note: You can execute ETL projects either from Sybase IQ InfoPrimer Development, or as a scheduled task.

Project Simulation

You can run an ETL project in simulation mode to monitor and validate your transformation process at every step.

Note: If you are running an ETL project in simulation mode and your objective is to test transformation rules, you may want to use a test data target.

In contrast to executing a project, simulation allows you to:

- Run projects that have unsaved changes.
- Monitor and validate the transformation process step by step. Data flow is visible on any link and within any component included. You can also inspect any component, and modify mappings and calculations.

  After making changes, you can reinitialize the component with the new settings and step to the next component without restarting the simulation.

During the final steps of a simulation, data is either written into the data sinks, or rolled back, depending on your selection. Many transformation components, such as the Data Calculator, allow you to change transformation rules and sample values during simulation, to validate your rule base for all potential content.

Note: You can simulate a project only after all components have been properly initialized.

ETL Components

ETL project components includes Source, Transformation, Lookup, Staging, Destination, and Loader components.

- Source components – deliver data for a transformation stream. A project typically starts with one or more source components. Source components have no input ports and at least one output port.
- Transformation components, Lookup components, and Staging components – apply specific transformations to the data in the transformation stream. These types of components have both input ports and output ports.
- Destination components (also called data sinks) – write data to specific targets. Destination components have one input port and no output ports.
- Loader components – extract and load data from a source database or file into the IQ database, without performing any transformation.

Component Properties Setup

Set the required properties before using any component in a project.

Use either:

- The Configuration window that appears when you add a component to the Design window, or,
- The Property window for a component added to the project.

Evaluating SBN Expressions

Set the Evaluate property for square bracket notation (SBN) expressions to be evaluated before a property value is used.

For some properties, the Evaluate property is selected by default.

You can change the Evaluate properties in either of these ways:

- In the Properties window, select the **Eval** option for the property or,
- Right-click a property and select **Evaluate.**

  If you select the **Eval** option for the Password property, the value appears as plain text.

See also

- *Square Bracket Notation* on page 178

Encrypting Properties

Encrypt the property values stored in the Sybase IQ InfoPrimer repository. Most entries in the repository are not encrypted, and are represented as readable character sets.
To toggle encryption properties, select the **Encrypt** option in the Properties window, or right-click the property and select **Encrypt**.

Referencing Property Variables

Reference properties having an associated variable in component expressions and procedures. The variable always contains the current value of a property evaluated, if Evaluate is set.

- To display the variable name, right-click the property in the Properties window and select **Reference Variable**.
- When setting up transformation rules in the JavaScript Debugger, click **Variable > Parameter** to access the Reference Variables.

See also

- *Drop-down Menus* on page 213

Adding and Removing Custom Properties

Custom properties incorporate variables that you can reference in component expressions or procedures.

1. Add custom properties to components. Components can contain expressions that are assigned in parameter sets or that are evaluated at runtime.
    1. In the Design window, select the component.
    2. In the Property window, right-click and select **Add.**
    3. Enter the name for the property. Do not use reserved JavaScript keywords. Inside the component, this property is referenced using the variable **REF.<name of property>**.
    4. In the **Prompt** field, enter a value to appear as the label for this property in the Properties window.
    5. In the **Description** field, enter a description to be used in the property tooltip.
    6. Click **OK.**
2. Remove the custom properties of a component and all references to the associated variables.
    1. In the Property window, right-click the custom property and select **Remove.**
    2. Click **OK** to confirm.

Adding Descriptions to Components

You can assign a name and a description to a component. The description and the name appear in the tooltip, which opens when you move your mouse over a component. The name also appears at the top of the component.

1. In the Design window, right-click the component and select **Description.**
2. Enter a name and description for the component. You can use HTML formatting tags to format the description.

Port Structure Configuration

All data in a project flows through component ports called IN-ports, which receive data, and output ports, which pass the data after processing. Each port owns the structure of the data flow. You can change the port structures of all components for which the structure is not directly dependent on component configuration. You can reference attributes added to the port structure immediately inside the component.

When connecting components, Sybase IQ InfoPrimer attempts to create a standard mapping between the output port and the input port. You can modify the mapping on a connection in the Mapping window. by right-clicking the connecting link, and selecting the **Mapping** option.

An unstructured port inherits the structure of a structured port, when a connection is added between them. You can add attributes to a port structure.

After you add a component to the project, the color of the component ports indicates the status of the component:

- Green – component is properly configured.
- Yellow – the port structure is defined but one or more mandatory properties are not defined.
- Red – no port structure is defined, but one or more mandatory properties are not defined.

Note: To enhance support for users with color disabilities, you can change the color of the yellow component port by selecting **Use enhanced color accessibility** in the **File > Preference** window.

Modifying Port Structures

Modify port structures using Structure Viewer window. If a port structure cannot be modified, the Edit Structure option is unavailable; you can only view the port structure using the View Structure option.

1. In the Design window, right-click the port and select **Edit Structure**.
2. In the Structure Viewer window, make the required changes, and click **Save.**

Adding Port Attributes

Add port attributes to all port structures participating in a selected connection.

1. In the Design window, right-click the port and select **Edit Structure**.
2. In the Structure Viewer window, select **Actions > Add**, or right-click the attribute and select **Add.**
3. Enter a name for the new attribute. The names for port attributes must start with an alphabet character and can contain only alphanumeric characters. Do not use reserved JavaScript keywords.
4. Select **Populate Attribute** to add the attribute to multiple port structures. The new attribute is added to all port structures participating in the selected connections and is automatically mapped. Click **OK.**

Deleting Port Attributes

Use the Structure Viewer window to remove port attributes.

1. In the Design window, right-click the port, and select **Edit Structure**.
2. In the Structure Viewer window, select **Actions > Remove**, or right-click the attribute and select **Remove.**

Modifying Port Attributes

Use the Structure Viewer window to modify port attributes.

1. In the Design window, right-click the port and select **Edit Structure**.
2. In the Structure Viewer window, change the attribute settings, and click **Save.**

Copying Port Structures from Other Ports

You can assign a port structure to a port based on any other available port in the current project.

1. In the Design window, select the port you want to assign a new structure to.
2. Right-click and select **Assign Structure > Copy Structure**.

You can copy the port structure from other ports of the same component. You can also copy any other port structure of the current project by selecting **Copy Structure**. If you select Copy Structure, a window displays an overview graph of the current project. You can select any of the available ports in the project.

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.
Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

Selecting a Port to Serve as the Source for your New Port Structure

Choose a source port for your new port structure.

1. Click the port you want to use as a source. The attribute structure of the selected port appears in the lower area of the window.
2. Click **Apply.**

Component Simulation

Run an ETL project in simulation mode to monitor and validate the transformation process step by step.

Data flow is visible on any link and within any component included. In simulation mode, many transformation components allow you to step through the current set of data and immediately visualize the result of any applied transformation.

Initializing Components

An ETL project can only be simulated after all its components have been properly initialized.
In the Design window, right-click the component, and select **Initialize**, or **Initialize and Step**.

If you modify an existing property setting of a component during simulation, you must reinitialize the component.

Stepping a Component Multiple Times

You can step a component multiple times during a simulation, to preview the behavior of a component with different property settings. Repetitive stepping does not increase the number of records delivered to the downstream component.

1. In the Design window, click the component.
2. In the Properties window, modify the transformation rules or property settings.
3. Right-click the component and select **Initialize.**
4. Right-click the component and select **Step.**
5. Go back to step 2.

When repetitively stepping a component, the same set of records at the input port are reprocessed in each step and forwarded without increasing the number of the records at the output port. You can step through many components from inside the component window, or by using the menu that appears when you right-click in the Sybase IQ InfoPrimer Development window.

Previewing Transformation Results

When working in simulation mode, you can divide the entire result set of the datasource (as defined by a query in the source or staging component) into data blocks, which contains a subset of records.

The number of records in each subset is related to the Read Block Size parameter, which appears in the Property window of the component. To enhance performance when stepping through the project, select a small number for the Read Block Size parameter.

1. Step through the project, including the component you want to preview.
2. Right-click the component, port, or connecting link, and select **Preview.**

   If a component has multiple ports, first select the port for which you are previewing transformation results.

Database Connection Settings

Specify database connection parameters in the Database Configuration window or Properties window.

- The Database Configuration window appears when you add a component to the Design window, or when you double-click an existing project component.
- The Properties window appears when you select a project component in the Design window.

The database connection parameters are:

- Interface – select the method or driver you want to use to connect to the destination or source database. To set special database options, click the Database options icon.

Note: Sybase Open Client™ must be installed on the same computer as Sybase IQ InfoPrimer Development, and the database server must be defined in the `%SYBASE%\ini\sql.ini` file on Windows, and in the `$SYBASE/interfaces` file on UNIX and Linux. If you are executing a project on an IQ InfoPrimer Server, the server must also have access to Open Client libraries.

The available interfaces for a component can be one or more of:

- Sybase
- ODBC – the ODBC driver must be installed on the same computer as Sybase IQ InfoPrimer Development, and a system datasource name (DSN) must be defined for the target. If the project is to be executed on an IQ InfoPrimer Server, the IQ InfoPrimer Server must also have access to the proper ODBC drivers and DSN.
- OLE DB
- Oracle
- DB2
- SQLite Persistent

- Host Name – select the source or destination database. The options that appear depend on the selected interface.

Note: If you selected the SQLite Persistent interface, you can enter an existing or new database file name.

- Database user name and password
- Database name
- Database schema – specify schema or owner to restrict the objects shown and create new tables in that schema.
- Standardize data formats – convert incoming date and number information into a standard format, which Sybase IQ InfoPrimer can move between systems that support different formats.
- Database options – override performance defaults and control the behavior of some transactions. For a list of database options, see Interface-specific database options.

See also

- *Connection Parameters* on page 513
- *Connecting to a SQLite Database* on page 521

Source Components

Source components deliver data for a transformation stream. A project typically starts with one or more source components. Source components have no IN-ports and at least one OUT-port.

DB Data Provider Full Load

DB Data Provider Full Load is a Source component that extracts data from any datasource that is accessible by an ODBC connection or native driver (DB2, Oracle, Sybase), or from

Microsoft SQL Server via OLE DB. The structure of the single output port mirrors the structure of the query result set.

Configuring a DB Data Provider Full Load Component

Add and configure a DB DataProvider Full Load component to your project.

1. Drag the **DB DataProvider Full Load** component onto the Design window. Alternatively, to open the configuration window, select the component in the Properties window, and click the **Properties** icon.
2. Enter the connection parameters. You must add a valid interface and host name.
3. Click the **Query** icon. Create and save a query to retrieve the data set from the datasource.

   You can create a query directly in the Query window, or click the **Query Designer icon** to open Query Designer and generate queries.
4. Click **Finish.**
5. In the Properties window, specify any other optional properties and database options.

See also

- *Data Provider Full Load Properties List* on page 73
- *Using the Query Designer to Create Queries* on page 188

Data Provider Full Load Properties List

The DataProvider Full Load properties list identifies connection parameters and other items you must define for the component.

Table 7. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Query	Create a query that retrieves information from the datasource. Use the Query window to create a simple query, or click the **Query Designer** icon to open the Query Designer.

Table 8. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.

Property	Description
Read Block Size	Determine the number of records retrieved by the component in a single step.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Running the DB Data Provider Full Load Demos

Sybase IQ InfoPrimer includes several demonstrations for the DB Data Provider Full Load component. These demos are available as Flash demos and as sample projects in the demo repository.

1. To run the flash demo, select **Help** > **Demonstrations** > **Source** > **DB Data Provider - Full Load**.

2. To access the sample projects, in the Navigator, select **Repository > TRANSFORMER.transformer. Repository > Projects**. Then select:
   - **Demo Transfer German Customers**
   - **Demo Transfer German Products**
   - **Demo Transfer German Sales**
   - **Demo Transfer US Customers**
   - **Demo Transfer US Products**

DB Data Provider Index Load

DB Data Provider Index Load is a Source component that performs incremental data loads based on an ascending index value. During execution, DB Data Provider Index Load ignores any previously extracted data records.

Use DB Data Provider Index Load to perform incremental loads that track source changes on a regular basis.

DB Data Provider Index Load has this impact on a simulation sequence:

- The Read Block Size value impacts the number of records loaded in a single simulation step.
- The value of the Load Index is updated in the repository when the project is executed during simulation, if **Execute Post-Processing as for successful execution** is selected. Load Index is not updated if you have selected **Execute Post-Processing as for failed execution**.

Configuring a DB DataProvider Index Load Component

Add and configure a DB DataProvider Index Load component.

1. Drag the **DB DataProvider Index Load** component onto the Design window. Alternatively, to open the configuration window, select the component in the Properties window, and click the **Properties** icon.
2. Add the connection parameters. You must add a valid interface and host name.
3. Click **Finish.**
4. In the Properties window, select an ascending index attribute from the list of database objects.
5. Click the **Query** icon. Create and save a query to retrieve the data set from the datasource.

   You can create a simple query directly in the Query window, or click the **Query Designer** icon.
6. In the Properties window, specify any other optional properties and database options.

See also

- *DB Data Provider Index Load Properties List* on page 76

Resetting the Ascending Index Value

You cannot directly manipulate the persistent value of load index in the repository. However, you can reset the value to the one stored in the Load Index Value property.
Right-click the project in the Navigator and select **Reset Execution Properties**.

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.
Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

DB Data Provider Index Load Properties List

DB Data Provider Index Load properties list identifies the connection parameters and other items you must define for the component.

Table 9. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.

Query	Create a query that retrieves information from the datasource. Use the predefined variable *LoadIndex* to qualify the selection criteria in the **WHERE** clause. Enclose *LoadIndex* with square brackets, because it is evaluated before the query is sent to the database, for example:
	`select * FROM SALES`
	`WHERE SA_DELIVERYDATE >'[LoadIndex]'`
	`ORDER BY SA_DELIVERYDATE`
	Note: Quote characters differ between database systems. In Microsoft Access databases, use # for datetime values.
	Use the Query window to create a simple query, or click the **Query Designer** icon to open the Query Designer.

Ascending Index	Select the attribute that contains the ascending index for delta load.

Table 10. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Load Index Value	The maximum value of the ascending index attribute is automatically used and stored when executing a Sybase IQ InfoPrimer job or schedule. The Load Index Value simulates the project using the specific value provided by the user. For example: `2005-01-19100`
Read Block Size	Determine the number of records retrieved by the component in a single step.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
Database Options	Set options that override performance defaults and control the behavior of some transactions.

Property	Description
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Simulating an Index Load

Simulate the incremental load when a project containing an Index Load component is fully configured.

1. To run the project, select **Run > Trace**. All records matching the condition specified by the Load Index Value property are processed.
2. Perform post-processing for a successful execution. Click **Yes.**
3. Simulate the project or the Index Load component again. The Index Load component does not return any records.

   Note: You can manually update the source table between step 1 and 2 to verify that the correct modified records are retrieved.
4. To simulate again, right-click the component and select **Reset Load Index Value**, or, in the Properties window, enter a new value for the Load Index Value property.

Running the Data Provider Index Load Demos

Sybase IQ InfoPrimer includes a demonstration for the Data Provider Index Load component. These demos are available as Flash demos and as sample projects in the demo repository.

1. To run the flash demo, select **Help > Demonstrations > Source > DB Data Provider – Index Load**.
2. To access the sample project, in the Navigator, select **Repository** > **TRANSFORMER.transformer.Repository** > **Projects**. Then select **Demo Transfer U.S. Sales on an incremental basis**.

Text Data Provider

Text Data Provider reads and transforms structured data from a text file into a table. The text source must contain fixed-length or delimited fields.

Configuring a Text Data Provider Component

Add and configure a Text Data Provider component.

1. Drag the **Text Data Provider** to the Design window.
2. In the Text Data Provider component window, select the text file to use as a datasource.

   Define the structural properties of data at the output port.

- File Content pane – displays the contents of the source document.
- Properties pane – the file description properties. You can modify the file description properties, if required.
- Output Port Content pane – a tabular view of data at the output port. In the Output Port Content pane, click **Regenerate the column definition icon to regenerate column definitions**. If you want the column names to be read from a data file, click the **Read column names from a data file** icon.

See also

- *Text Data Provider Properties List* on page 80
- *Reading Column Names from a Data File* on page 79

Reading Column Names from a Data File

Use the Text Data Provider component window to read the column names from a data file.

1. In the Output Port Content pane of the Text Data Provider component window, click the **Read column names from the data file** icon.
2. Provide the line number of the record that contains the column heading. Click Enter to confirm.

   You can double-click the column headings to edit the name of the column.

Skipping Row Delimiters

The line containing the column headings is not automatically skipped when the data is processed.
To skip a specified number of rows at the beginning of the input tables for a load, enter a value in the Skip First Rows field.

If the rows to be skipped do not contain the same format as later rows; for example, if there is a “header” row with no column delimiters, while later rows all contain five columns, it is still counted as one row.

Note: Quoted row delimiters are not regarded as row delimiters.

Delimiter Considerations

Considerations for using quote characters, row and, column delimiters.

- Quotes can appear anywhere within a value.
- A file and each value must have an even number of quote characters.
- Quotes cannot be escaped.
- Delimiters (row or column) within quotes are treated as a part of the value.
- Quotes are always stripped from data values.

For a value 'abc,def', these examples are quoted correctly, and are read as one value:

- "abc,def"
- abc","def
- "a"bc","d"e"f
- abc",def"
- """abc,"def"""""""

These examples are read as two values:

- abc,def
- ""abc,def""
- "abc",def

This example is read as an error: "'abc,def".

Text Data Provider Properties List

Text Data Provider properties list identifies items about the structure of source files. Properties are initially set when you add the component to the project.

Table 11. Required Properties

Property	Description
Text Source	Identify the text file to use as the datasource. You can select the datasource when you add Text Data Provider to a project, or from the Properties window. To select a datasource from the Properties window, click the **Text Source** icon, then select the file.
Columns	Define columns for the data in your source file.

Table 12. Optional Properties

Property	Description
Row Delimiter	Specify how each row is delimited: • Position (fixed-line position) • LF (line feed) • CR (carriage return) • CRLF (carriage return followed by a line feed) You can enter a different delimiter.
Row Length	Specify the number of characters in each fixed row, if you have selected Position as the Row Delimiter.

Property	Description
Column Delimiter	Specify how columns are delimited: • Position (fixed column positions) • Tab • Comma • Semicolon Enter a different delimiter.
Column Quote	Specify how the values in the source file are quoted: • None • Single quote • Double quote Eenter a different quote character or string.
Fixed by Bytes	Specify how to interpret the values provided for line length, column start, and column end: • Not selected (default) – values are interpreted as number of characters. • Selected – the values are interpreted as number of bytes. For example, suppose your source file includes **abcÖÐÎÄ abcdef** and has these characteristics: • File Type – Fixed (Variable Line) • Encoding – GB2312 • Row delimiter – '\n' • Column definition – column1: 1-7; column 2: 9-10 If you select the Fixed by Bytes option: • Column 1 displays the first 7 bytes, **abcÖÐÎÄ** . • Column 2 displays the 9th and 10th bytes, **ab** . If you do not select the Fixed by Bytes option: • Column 1 displays the first 7 characters, **abcÖÐÎÄ a** . • Column 2 displays the next 2 characters, **cd** .
Null Byte Substitute	Set the character to replace null bytes.
Skip Rows	Skip a specified number of rows in the row sequence.
Encoding	Set the current character encoding.

Property	Description
Support Unicode	Set Unicode support.
Read Block Size	Determine the number of records retrieved by the component in a single step.
Read empty as NULL	Replace values read as “empty” with “null.” Set this option if the target is a database table and the text file does not provide values for all attributes that must not be empty but can accept NULLs.

Running the Text Data Provider Demos

Sybase IQ InfoPrimer includes several demonstrations for the Text Data Provider component. These demos are available as flash demos and as sample projects in the demo repository.

1. To run the flash demo, select **Help > Demonstrations > Source > Text Data Provider**.
2. To access the sample projects, in the Navigator, select **Repository > TRANSFORMER.transformer.Repository > Projects**. Then select:
   - **Demo Transfer German Customers**
   - **Demo Transfer German Sales**
   - **Demo Transfer U.S. Customers**

XML Via SQL Data Provider

The XML via SQL Data Provider component loads hierarchical XML data into a relational schema that you can query like a relational database.

XML via SQL Data Provider is designed for data-centric XML documents, such as sales order, stock quotes, or scientific data, which are characterized by a regular hierarchical structure.

Configuring an XML Via SQL Data Provider Component

Add and configure an XML Via SQL Data Provider component.

1. Drag the XML via SQL Data Provider component onto the Design window.
2. In the Properties window, click the **XML Source** icon and select the XML to use as a datasource. You can specify **HTTP**, **FTP**, **URL**, or file name.
3. Click the **Data Output** icon.
4. Click the **Properties** icon to open the XML Port Manager. Specify a query for each output port.

   By default, XML via SQL Data includes one output port, but you can add ports. Any output ports you add in XML Port Manager appear in the Design window.

See also

- *Working with XML Port Manager* on page 83

Updating the Output Port Structure

Update the output port structure to reflect changes made to the XML source file.
Right-click the XML via SQL Data Provider component and select **Reconfigure**.

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the XML source file, and applies the updates to the output port structure.

Working with XML Port Manager

The XML Port Manager window lets you create queries against the XML source file, to define one or more output data streams.

- XML source view – displays the contents of the source document.
- Data Model tab – displays a relational view of the source document.
- Reference tab – displays the available component variables.
- Output port area – used to write queries against the data model, and send the results to a particular output port. Although XML Port Manager is, by default, configured with one output port, you can add ports, and write additional queries.

Writing Queries

When you open XML Port Manager, the output port area includes a standard query against the XML view, which returns all columns and all rows to OUT1.

Assume that your XML source document contains customer data expressed as attribute values of each data node:

```
<root>
    <data id="101" fname="Michaels" lname="Devlin"
        address="114 Pioneer Avenue" city="Kingston"
        state="NJ" zip="07070"/>
    <data id="102" fname="Beth" lname="Reiser"
        address="33 Whippany Road" city="Rockwood"
        state="NY" zip="10154"/>
     <data id="103" fname="Erin" lname="Niedringhaus"
        address="190 Windsor Street" city="Tara"
        state="PA" zip="19301"/>
</root>
```

To retrieve customer attributes against the XML, you can use a query similar to:

```
select * from V_XML_CONTENT WHERE TAB_data_ATT_city = 'Kingston'
```

This query opens the Content Browser, and returns only those rows for which the `city` value matches `Kingston`. In the tabular view, you can write a query that looks similar to:

```
select * from TAB_data where ATT_city='Kingston'
```

Note: XML data relationships are generally expressed as parent/child, or as node/attribute relationships. The Content Browser returns XML Port Manager query results as columns and rows. Column and row references refer to query results, not XML data.

Retrieving Data from Your XML Data Source

Extract data from an XML datasource using the XML Port Manager.

Write your queries directly into the port field in the output port area using the standard SQL syntax:

```
select column
FROM table_name
```

- Query Designer helps you design queries for the tabular view. Depending on the structure of the XML source, you may need to create joins between the tables to return rows of data.
- The default XML view_name is `V_XML_CONTENT` . To return a row, qualify the **select** statement with a **WHERE** clause (`select * from V_XML_CONTENT WHERE TAB_state_ATT_state = 'NY'`).
- In the tabular view, XML nodes formatted as attribute expressions sometimes create a wrapper element you can qualify with a **WHERE** clause (`select * from TAB_data where ATT_city='Kingston'`) to return a row.

Writing Queries Against the Table View

Use the XML Port Manager or the Query Designer to write queries.

You can write queries against the Table view directly into the port field in the output port area, or use the Query Designer. Use standard SQL syntax to query tables in the Table view.

Adding and Removing Ports

Add and delete ports.

Right-click the port section and select **Add Port** or **Remove Port**.

Setting Up a Sample Project

Set up the XML via SQL Data Provider component using a simple example. To follow this example, use the `PRODUCTS.xml` as the XML source; it is located in the `Demodata` subdirectory of the Sybase IQ InfoPrimer installation directory.

XML Port Manager

Define the ports in the output port area of the XML port manager. Each port is described by a **select** statement based on the XML data model tables.

XML Source

A simple product structure where each product is described with an ID (`PR_ID`), name (`PR_NAME`), product group (`PR_GROUP1`), and price (`PR_PRICE`).

```
<?xml version="1.0" encoding="UTF-8"?>
    <dataroot xmlns:od="urn:schemas-solonde-com:demodata"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="PRODUCTS.xsd"
        generated="2005-01-24T16:13:26"><PRODUCTS>
        <PR_ID>435672</PR_ID>
        <PR_NAME>24 CD Rom Drive</PR_NAME>
        <PR_GROUP1>CD Rom</PR_GROUP1>
        <PR_PRICE>134</PR_PRICE>
    </PRODUCTS>
    <PRODUCTS>
        <PR_ID>435673</PR_ID>
        <PR_NAME>Notebook 235</PR_NAME>
        <PR_GROUP1>Notebook</PR_GROUP1>
        <PR_PRICE>1455</PR_PRICE>
    </PRODUCTS>
</dataroot>
```

The Data Model

Tha data model includes one table for the root element (`TAB_dataroot`), followed by one or more tables for elements at level 1.

In this example, only one element at this level exists (`TAB_PRODUCTS`). At the next level, you find a table for each element on level 2 (`TAB_PR_ID, TAB_PR_NAME`, `TAB_PR_GROUP1`, `TAB_PR_PRICE`). There can be more nested levels in the XML document, and each level creates another set of tables.

Note: You can change the prefixes for the generated table names in the `DB Schema Options` property.

Root Level	Elements lLvel 1	Elements Level 2

`TAB_dataroot`
`ATT_ROW_ID`
`ATT_FK_generated`
`ATT_xmlns_od`
`ATT_xsi_no`
	`TAB_PRODUCTS`
	`ATT_ROW_ID`
	`ATT_FK_dataroot`
		`TAB_PR_ID`
		`ATT_ROW_ID`
		`ATT_FK_PRODUCTS`
		`ATT_PR_ID`
		
		`TAB_PR_NAME`
		`ATT_ROW_ID`
		`ATT_FK_PRODUCTS`
		`ATT_PR_NAME`
		
		`TAB_PR_GROUP1`
		`ATT_ROW_ID`
		`ATT_FK_PRODUCTS`
		`ATT_PR_GROUP1`
		
		`TAB_PR_PRICE`
		`ATT_ROW_ID`
		`ATT_FK_PRODUCTS`
		`ATT_PR_PRICE`

The tables are linked through foreign keys. Table `TAB_PRODUCTS` is linked to `TAB_dataroot` through the `ATT_FK_dataroot` attribute.

The tables at level 2 are linked to table `PRODUCTS` through the `ATT_FK_PRODUCTS` attribute. To create the view containing the `PRODUCTS` records, the tables at level 2 must be joined with the `TAB_PRODUCTS` table.

The join is qualified by the `ATT_FK_PRODUCTS` attribute of each level 2 table and the `ATT_ROW_ID` of `TAB_PRODUCTS`. The only selected attributes are the value attributes of level 2 tables: `ATT_PR_ID`, `ATT_PR_NAME`, `ATT_PR_GROUP1` and `ATT_PR_PRICE`.

```
select  TAB_PR_ID.ATT_PR_ID,
TAB_PR_NAME.ATT_PR_NAME, TAB_PR_GROUP1.ATT_PR_GROUP1,
TAB_PR_PRICE.ATT_PR_PRICE
FROM  TAB_PRODUCTS, TAB_PR_ID, TAB_PR_NAME,
TAB_PR_GROUP1, TAB_PR_PRICE
WHERE  TAB_PR_ID.ATT_FK_PRODUCTS =
TAB_PRODUCTS.ATT_ROW_ID AND
TAB_PR_NAME.ATT_FK_PRODUCTS = TAB_PRODUCTS.ATT_ROW_ID
AND  TAB_PR_GROUP1.ATT_FK_PRODUCTS =
TAB_PRODUCTS.ATT_ROW_ID AND
TAB_PR_PRICE.ATT_FK_PRODUCTS =
TAB_PRODUCTS.ATT_ROW_ID
```

XML via SQL Data Provider Properties List

XML via SQL Data Provider Properties list identifies sets processing options for the XML source file.

Table 13. Required Properties

Property	Description
XML Source	Identify the datasource. You can select the XML datasource when you add a component to a project, or select the file from the Properties window. To select a datasource from the Properties window, click **XML Source**, then select the file.
Data Output	Open the XML port manager, a management console where you can query XML source.

Table 14. Optional Properties

Property	Description
Document Schema	Identify an external schema (`.xsd`) or DTD that you can use to validate the XML source.
Namespace Schema	Point to the location of an external namespace schema. An XML schema consists of components such as type definitions and element declarations that you can use to assess the validity of well-formed element and attribute information items.

Property	Description
Validate Schema	Enable schema and DTD validation.
XML Options	• Full schema check – set to 1 to check for items that may be time consuming or memory intensive. Particle unique attribution constraint checking and particle derivation restriction checking are controlled by this option. The default value is 0. • Ignore external DTD – set to 1 to ignore an external DTD referenced within the document. The default value is 0. • Process namespace – set to 0 to not consider namespace specification during parsing. The default value is 1. • Preserve Element whitespace – set to 1 to preserve white space in the XML element value. Set to 0 to trim white space from the XML element values. If the XML element value includes only white space and the value is set to 0, it is interpreted as an empty element value.
DB Schema	Select the database schema setup (create tables) script. Use this option to enforce a fixed data model.

Property	Description
DB Schema Options	Customize the settings for tables and attributes generated from the XML structure, including the prefixes for table and attribute names. The DB Schema options are: • Attribute name case – formats the attribute names generated from the XML. The values “upper” and “lower” convert attributes names accordingly. “Mixed” (the default) leaves the names as they appear in the XML document. • Attribute name prefix – prefix for every generated attribute name. • Create indexes – set to1 (the default) to automatically generate indexes on the primary keys of the tables. • Create flat views – set to1 (the default) to automatically generate a view called V_XML_CONTENT. This view joins all tables and returns all XML data in a broad table. If the database schema results in more than 32 tables, this view does not work and the option has to be switched off. • Foreign key prefix – prefix for attributes that are foreign keys. • Ignore Empty Leaf Element Values – set to 1 to not include column entries for specific XML leaf elements that contain no data. If set to 0, the database created from the XML document includes column entries for all XML leaf elements whether empty or not. • Primary key name – attribute name to be used for primary keys. • Table name case– formats the table names generated from the xml. The values “upper” and “lower” convert the table names accordingly. “Mixed” leaves the names as they appear in the XML document. • Table name prefix – prefix to be used for generated table names
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Read Block Size	Determine the number of records retrieved by the component in a single step. If the component has more than one output port, the read block size is ignored and the component provides data at all ports, in a single step.

Running the XML Via SQL Data Provider Demos

Sybase IQ InfoPrimer includes a demonstration for the XML via SQL Data Provider component. These demos are available as Flash demos and as sample projects in the demo repository.

1. To run the flash demo, select **Help > Demonstrations > Source > XML via SQL - Data Provider**.
2. To access the sample projects, in the Navigator, select **Repository > TRANSFORMER.transformer.Repository > Projects**. Then select **Demo XML via SQL Data Provider**.

CDC Provider Sybase Replication Server

The CDC (Capture Data Changes) Provider Sybase Replication Server component is used for implementing incremental load.

The CDC Provider Sybase Replication Server component:

- Receives data changes from Sybase Replication Server, translates them to a standard data flow, and then sends the data out to the next component.
- Automates configuring Sybase Replication Server, which includes marking the source table as replicated, creating or dropping replication definitions, connections, and subscriptions. This process is called Creating or Dropping Replication, and it enables Replication Server to start or stop capturing source table data changes.

Note: The CDC Provider Sybase Replication Server component supports only Adaptive Server Enterprise and Oracle as the source database.

See also

- *Updating the Interfaces Files* on page 90
- *Configuring Replication CDC Service Name for Each Sybase IQ InfoPrimer Server* on page 91
- *Configuring Oracle as a Replication Source* on page 92

Getting Started with CDC Provider Sybase Replication Server Configuration

Identify the tasks to perform before configuring the CDC Provider Sybase Replication Server component.

1. Install Replication Server.

   Note: You need not install Replication Server and Sybase IQ InfoPrimer Server on the same machine.

2. Allocate sufficient disk space for the Replication Server partition. See Replication Server Commands in the *Replication Server 15.5 Reference Manual*

3. Configure the Replication CDC Service:
   a) Edit the `interfaces` files for the source database, Replication Server, and Sybase IQ InfoPrimer Server.
   b) Configure the Replication CDC Service Name for each Sybase IQ InfoPrimer Server.
4. If the source database is Adaptive Server Enterprise, use **rs_init** to add the source database to the Replication Server. See the *Replication Server Configuration Guide*.
5. If the source database is Oracle, add Oracle to the Replication Server.

Updating the Interfaces Files

Use a text editor to modify all the `interfaces` file for the souce database, Replication Server and Sybase IQ InfoPrimer.

1. Navigate to the `interfaces` file.
   - On Windows, the file is `<installation_directory>\ini\sql.ini`.
   - On UNIX and Linux, the file is `<installation_directory>/interfaces`
2. Modify all three `interfaces` files to include:
   - Entries for Replication CDC Services of all Sybase IQ InfoPrimer Servers.
     - On Windows for example, use:
       ```
       [<cdc_service_name>]
       master=TCP,<machine_name>,<port>
       query=TCP,<machine_name>,<port>
       ```
     - On UNIX or Linux, use:
       ```
       <cdc_service_name>
       master tcp sun-ether <machine_name> <port>
       query tcp sun-ether <machine_name> <port>
       ```

     where:

     <*cdc_service_name*> is a unique Replication CDC Service name to be used by a single Sybase IQ InfoPrimer Server.

     <*machine_name*> is the name of the machine on which the grid Sybase IQ InfoPrimer Server runs.

     <*port*> is the port on which the Replication CDC Service listens.
   - A SYBETL_VIR_RDBMS entry that contains all CDC Service entries.
     - On Windows for example, use:
       ```
       [SYBETL_VIR_RDBMS]

           master=TCP,<machine_name_1>,<port>

       query=TCP,<machine_name_1>,<port>

       master=TCP,<machine_name_2>,<port>

       query=TCP,<machine_name_2>,<port>
       ```
     - On UNIX and Linux, use:
       ```
       SYBETL_VIR_RDBMS
       ```

```
master tcp sun-ether <machine_name_1> <port>
```

```
query tcp sun-ether <machine_name_1> <port>
```

```
master tcp sun-ether <machine_name_2> <port>
```

```
query tcp sun-ether <machine_name_2> <port>
```

Note: The IP port of the virtual database and Replication CDC Services should match.

- If the source database is Adaptive Server Enterprise, create or copy the entries for your Adaptive Server Enterprise source database, Replication Server, and Embedded Replication Server System Database (ERSSD) or RSSD.

  If the source database is Oracle, create or copy the entries for your Replication Agent™, Replication Server, and Embedded Replication Server System Database (ERSSD) or RSSD.

  - On Windows for example, use:

    ```
    [<ase_name>]
    master=tcp,<ase_machine_name>,<ase_port>
    query=tcp,<ase_machine_name>,<ase_port>

    [<repserver_name>]
    master=tcp,<repserver_machine_name>,
           <repserver_port>
    query=tcp,<repserver_machine_name>,
           <repserver_port>
    [<erssd_name>]
    master=tcp,<erssd_machine_name>,<erssd_port>
    query=tcp,<erssd_machine_name>,<erssd_port>
    ```

  - On UNIX and Linux, use:

    ```
    <ase_name>
    master tcp sun-ether <ase_machine_name> <ase_port>
    query tcp sun-ether <ase_machine_name> <ase_port>

    <repserver_name>
    master tcp sun-ether <repserver_machine_name>
    <repserver_port>
    query tcp sun-ether <repserver_machine_name>
    <repserver_port>
    <erssd_name>
    master tcp sun-ether <erssd_machine_name> <erssd_port>
    query tcp sun-ether <erssd_machine_name> <erssd_port>
    ```

Configuring Replication CDC Service Name for Each Sybase IQ InfoPrimer Server

Define the Replication CDC Service name.

- Use the **repcdcinstancename** command line parameter – if you define this parameter value, the grid engine ignores the Replication CDC Service configurations in the `svc.conf` file, and uses the parameter value to start the service. To start a grid engine with the Replication CDC Service name, ETL_RCS_INS1, using the command line, enter:

  ```
  GridNode --repcdcinstancename ETL_RCS_INS1
  ```

Note: The CDC Provider Sybase Replication Server component does not work if Replication CDC Service is not properly configured and running.

- Update the `svc.conf` file:

  1. Navigate to the `etc` directory of the installation folder and use a text editor to open the `svc.conf` file.
  2. Update *instance_name* to include the Replication CDC Service Name. For example, if your Replication CDC Service name is ETL_RCS_INS1, enter:

     Note: All grid engines that start the Replication CDC Service must be in the same subnet.

     ```
     repcdc {
       type = "repcdc";
       container = "inprocess";
       autostart = true;
             config {
                         instance_name = "ETL_RCS_INS1";
                    }
             }
     ```

     Note: The Replication CDC Service name must be unique for each grid engine.
  3. Save the file as UTF-8 encoded. Otherwise, the grid engine cannot read the file, and may not start.

Configuring Oracle as a Replication Source

Configure Oracle as a replication source. If you have already configured the Oracle database in your replication environment, you must still, to use Sybase IQ InfoPrimer, configure the Oracle instance, configure Replication Agent, and add the primary database to the replication system.

Installing Replication Agent (RAX) and Creating a Replication Agent for Oracle (RAO) Instance

Install Replication Agent and create an instance.

1. Install Replication Agent.
2. Copy the downloaded license file to `SYSAM-2_0\licenses` in the installation folder.
3. Start an RAO instance. For example:

   ```
   ra_admin -c rao_inst1 -p 1333 -t oracle
   ```
4. Include the Oracle JDBC™ driver `jar` files in the CLASSPATH variable. For example:

   ```
   set CLASSPATH=%CLASSPATH%;C:\oracle\product
   \10.2.0\db_2\jdbc\lib\ojdbc14.jar
   ```

Configuring the Oracle Instance

Connect to the Oracle instance as a system administrator, using **SQLPLUS**,

1. Prepare Oracle to use redo logs.

   To verify the archive log mode, enter:

   ```
   select log_mode from v$database;
   ```

   If the archive log mode is on, you see:

   ```
   LOG_MODE
   -------
   ARCHIVELOG
   ```

   If the archive log mode is off,you see:

   ```
   shutdown immediate;exit
   ```

   Run the **.sqlpus/nolog** command to set the archive log mode to on, then enter:

   ```
   connect sys/password as sysdba;
   startup mount;
   alter database archivelog;
   alter database open;
   alter system set recyclebin=off;
   ```
2. Enable supplemental logging for the source table:

   ```
   ALTER TABLE T1 ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
   ```

   Note: You must enable supplement logging for the source table.
3. Add primary key information to the Oracle redo log:

   ```
   alter database add supplemental log data (primary key,
   unique index) columns;
   select SUPPLEMENTAL_LOG_DATA_MIN,
   SUPPLEMENTAL_LOG_DATA_PK,
   SUPPLEMENTAL_LOG_DATA_UI from v$database;
   ```

   If the primary key information is successfully added, you see:

   ```
   SUPPLEME SUP SUP
   -------- --- ---
   YES      YES YES
   ```
4. Create Oracle users for Replication Agent and Replication Server, and then provide permission to them:

   ```
   grant connect, resource,dba
   ```

Note: Do not use the Oracle user for Replication Server to perform any DML transactions, as Replication Server does not capture data changes made by this user.

Configuring Replication Agent

Start the RAO instance and connect to it using **isql**.

1. Set the archive log file path of the source Oracle database. Enter:

```
ra_config pdb_include_archives, true
go
ra_config pdb_archive_path, <path-to-oracle-archive-directory>
go
```

2. Configure a connection between Replication Agent and the primary database. Enter:

```
ra_config pds_host_name, <the host name of the source oracle>
go
ra_config pds_port_number <the port number of the source oracle>
go
ra_config pds_database_name,<the source oracle database name>
go
ra_config  pds_username, <the oracle user for Replication Agent>
go
ra_config pds_password, <password>
go
test_connection PDS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
PDS  succeeded
```

3. Configure the Replication Agent connection to Replication Server. Enter:

```
ra_config rs_host_name, <the host name of the Replication Server>
go
ra_config rs_port_number, <the port number of the
Replication Server>
go
ra_config rs_username, <the Replication Server user for
Replication Agent>
go
ra_config rs_password, <password>
go
ra_config rs_source_ds <the current RAO instance name>
go
ra_config rs_source_db, <the source oracle database name>
go
```

Note: You must specify the RAO instance name.

4. Configure the Replication Agent connection to the ERSSD:

```
ra_config rssd_host_name <the host name of the ERSSD>
go
ra_config rssd_port_number, <the port number of the ERSSD>
go
```

```
ra_config rssd_username, <the ERSSD user for Replication Agent>
go
ra_config rssd_password, <password>
go
ra_config rssd_database_name, <the database name of the ERSSD>
go
test_connection RS
go
```

If the connection is established successfully, you see:

```
Type Connection
---- ----------
RS succeeded
```

5. If the charset of Replication Server is not the same as Replication Agent, enter:

   ```
   ra_config rs_charset, <the charset of the Replication Server>
   ```

6. To correctly handle update or delete transactions, configure **ltl_send_only_primary_keys** as false:

   ```
   ra_config ltl_send_only_primary_keys, false
   ```

   Note: Sybase IQ InfoPrimer cannot handle delete or update transactions if you do not perform this step.

7. Initialize Replication Agent:

   ```
   pdb_xlog init
   ```

Adding Primary Database to the Replication System

Connect to Replication Server using **isql.**
Enter:

```
create connection to  < rs_source_ds >.< rs_source_db >
set error class rs_sqlserver_error_class
set function string class rs_sqlserver_function_class
set username <the oracle user for Replication Server>
set password <password>
with log transfer on, dsi_suspended
```

where:

- *rs_source_ds* – is the same as in Replication Agent.
- *rs_source_db* – is the same as in Replication Agent.

Resuming Replication Agent

Resume Replication Agent, when it is ready to replicate transactions.

1. In Replication Agent, execute:

   ```
   resume
   go
   ```

2. Verify if Replication Agent is replicating:

   ```
   ra_status
   go
   ```

If Replication Agent is set up correctly, you see:

```
State        Action
----------- ------------------------
 REPLICATING  Ready to replicate data.
```

Configuring a CDC Provider Replication Server Component

Add and configure a CDC Provider Replication Server component.

1. Drag the CDC Replication Server Provider component onto the Design window.
2. Specify the name of the Replication Server to use to capture data changes, along with its user name, and password.
3. Set the replication definition options in the Source Table Options field.
4. To filter qualified data changes, select one of these capture modes:
   - Full – receives all changes and sends them one by one. The output changes are same as the input changes from Replication Server.
   - Last – sends only the last received data changes of each row. All changes with the same key values are merged into a single change.

     For example, if there are two changes:

     ```
     1.update test_table set col_1='x' where key_col='A'
     2.update test_table set col_1='y' where key_col='A'
     ```

     and if you selected Last as the capture mode, the output is:

     ```
     update test_table set col_1='y' where key_col='A'
     ```

   Sybase recommends that you set this property to Last, which is appropriate for most incremental loading. If set to Last, the Insert and Update changes are regarded as Upsert changes.

   Note: If you select Full, and if there are several changes on the same row, you must ensure the sequence while loading to the target database.
5. If capture mode is set to Last, enter the Stage Mode property to specify whether you want the component to stage all received data changes in memory, or in IQ.

   Note: For a 32-bit grid engine, Sybase recommends that you set the stage mode to IQ.
6. Specify the staged IQ configurations in the **IQ for Stage Mode Options** field, if the Capture Mode is set to Last, and the Stage Mode is set to IQ.
7. In the Ports Options field, specify the output ports details.
8. Specify any other optional properties.

See also

- *CDC Provider Replication Server Properties List* on page 98
- *Setting Replication Definition Options* on page 97

- *Configuring the Output Ports* on page 97
- *Creating Replication* on page 97

Setting Replication Definition Options

Set the replication definition in the CDC Configuration window.

1. Click the **Source Table Options** icon.
2. Enter connection details for the source database.
3. Click **Options** to select the source table to replicate.
4. Select the **Replicate** option for each column to replicate.
5. Select the **Key** option against the column you want to mark as the primary one.

   You must select one or multiple columns as the key column to prevent errors during replication creation.
6. Click **Save.**

Creating Replication

Creating replication includes creating the replication definition, replication connections, function strings, and replication subscriptions with “No materialization” as the materialization method. It enables Replication Server to start capturing the source table data changes.

1. Right-click the component and select **Create Replication**.
2. Confirm that source and target tables are synchronized and click **Yes.**

   On successful creation, the Replication property in the Property window changes to “Created,” and the status is written to the repository.

Dropping Replication

Dropping replication includes dropping the replication definition, replication connections, function strings, and replication subscriptions. It enables Replication Server to stop capturing the source table data changes, and clears all unprocessed data changes of the source table.

1. Right-click the component and select **Drop Replication**.
2. Confirm that you are dropping the replication. Click **Yes.**

   The Replication property in the Property window changes to “Dropped,” and the status is written to the repository.

Configuring the Output Ports

Configure the output ports in the CDC Provider Ports window.

1. Click the **Ports Options** icon.
2. Click the **Add Port** icon, and enter a name for the new port. Click **OK.**

Note: You cannot change a port name in the CDC Provider Ports dialog. To change the port name, go to the Design window, right-click the port, and select **Description.**

3. To change the output data from an output port, select an option from the **Function** drop-down list, and click **OK.**
4. To remove a port, select it and click the **Remove Port** icon. There must be at least one output port on the CDC Provider Replication Server component.

CDC Provider Replication Server Properties List

The CDC Provider Replication Server Properties list identifies the required and optional properties of the component.

Table 15. Required Properties

Property	Description
RepServer Name	Specify the Replication Server to use to capture data changes. After Replication is created, this property cannot be modified. To modify it, right-click the component and select **Drop Replication**.
Rep Database	Object name generated during replication creation. This property is read-only.

Property	Description
Source Table Options	Set the replication definition options: • Interface – identify the method or driver to use to connect to the primary datasource. • Host Name – identify the primary datasource. The options depend on the selected interface. • User and Password – identify an authorized database user, and protect the database against unauthorized access. If the source database is Adaptive Server Enterprise, the database user requires “sa” or “dbo” permission or “replication_role.” • Database – identify the database to use as datasource. Also select an appropriate interface, and in some cases, specify an appropriate user ID and password. • Schema – identify the schema or owner to use as datasource. The objects that appear are restricted accordingly and new tables are created in that schema. • Table – identify the source table. • Replicate – identify the column to be replicated. **Note:** Since large object (LOB) data processing is not supported, a column with LOB type is not replicated. • Key – identify the columns with keys. Replication Server and Sybase IQ InfoPrimer use the key to identify each row of the source table. **Note:** Replication Server does not allow using column with LOB type as key. The key values should be unique for each row and the key columns must be replicated. • Size – identify the column size. If the source database is Oracle, enter Replication Agent details: • RepAgent Host – Replication Agent name, which must be the same as in Replication Agent. • RepAgent Database – Replication Agent database name, which must be identical to the configuration value of ‘ **rs_source_db** ’ in Replication Agent. • RepAgent Username and Password – user name and password to connect to Replication Agent. After Replication is created, you cannot modify this property. To modify it, right-click the component and select **Drop Replication**.

Property	Description
Port Options	Specify the port type of each output port: • Port Name – specify the output port name. • Function – specify the data changes to be sent from this port: • Insert – sends insert changes. • Delete – sends delete changes. • Update – sends update changes. • Upsert – sends insert and update changes. • All – sends all data changes.
Capture Mode	Specify whether to send all received data changes or only the last received data changes on each row, to the next component. By default, this property is set to "Last," which is appropriate for most incremental loading.
Replication	Indicates whether replication is created or dropped. The value for this property is generated by Sybase IQ InfoPrimer and is read-only.

Table 16. Optional Properties

Property	Description
Rep Server User and Password	Specify an authorized Replication Server user name and password.
Stage Mode	Specify whether data changes should be staged in the memory or in an IQ temporary table. If the capture mode property is set to Last, provide a value for the stage mode property. **Note:** For a 32-bit grid engine, Sybase recommends that you set stage mode to IQ.
IQ for Stage Mode Options	Specify configuration for staging IQ. This property is enabled only when capture mode is Last, and stage mode is IQ.
Timeout seconds for no data	Specify the wait time, in seconds, for the component before it stops waiting to receive data changes from Replication Server.
Read Block Size	Determine the number of records retrieved by the component in a single step.

Property	Description
Auto Initial Load	Specify whether you want the initial load to take place automatically. Before transferring source table data changes, Sybase IQ InfoPrimer must perform an initial load to synchronize the source and destination table data, and create replication objects to make Replication Server start capturing data changes. **Note:** Since Replication Server does not support materialization for Oracle, this property does not work if the source database is Oracle. While executing or simulating a project: • If this property is selected and Replication is not created, Sybase IQ InfoPrimer creates the replication definition, connection, and function strings, and uses the atomic materialization method to create the subscription. This materialization method locks the source table until all source table records are captured by Replication Server as insert operations in a transaction. After this, simulation or execution performs the initial load. • If this property is not selected and Replication is not created, you see an error message. • If this property is selected and execution or simulation fails, right-click the component and select **Drop Replication**. Try to execute or simulate the project again. If you right-click the component and select **Create Replication** while this property is selected, initial load does not take place. **Note:** While simulating a project, if this option is selected and you click **Reset** and **Start** to restart simulation, the CDC Provider Sybase Replication Server component does not receive the source table initial data a second time. Replication Server pushes initial data to Sybase IQ InfoPrimer only once.
Output Old Value	Select to output old column values of each data change. After Replication is created, you cannot modify this property. To modify it, right-click the component and select **Drop Replication**.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Property	Description
Save Interval	Specify the amount of time, in minutes, for which Replication Server should save the data changes after it is received by Sybase IQ InfoPrimer. By default, Replication Server discards each data change after Sybase IQ InfoPrimer receives it. Set this property to have Replication Server save data changes for a while after Sybase IQ InfoPrimer receives them. You must save data changes, since a project or job failure can cause the destination table changes to roll back, if project level transactionality is enabled. In this case, all source table data changes received are not applied to destination tables. In subsequent executions, the data changes that are not processed successfully in the previous execution, that still exist in the Replication Server, are sent to the CDC Replication Server Provider component, along with any new changes. This ensures that the destination tables are updated correctly. If the save interval is too short, some data changes from running a failed project are not sent to the destination components, causing the destination tables to be out of sync with the source table data. If the save interval is too long, duplicate data changes are sent to the component, leading to decreased performance, and increased resource usage.

Transformation Components

Transformation components have both input ports and output ports and apply specific transformations to the data in the transformation stream.

See also

- *Character Mapper* on page 102
- *Copy Splitter* on page 105
- *Data Calculator JavaScript* on page 106
- *Data Splitter JavaScript* on page 110
- *SQL Executor* on page 113

Character Mapper

The Character Mapper is a Transformation component that replaces characters and strings in an input record. The Character Mapper applies replacement mapping to all but selected attributes.

Use the Character Mapper to replace characters or strings, for example, to change a German umlaut (ä) to ae or Unicode characters.

Configuring a Character Mapper Component

Add and configure a Character Mapper component.

1. Drag the **Character Mapper** component onto the Design window.
2. Link the input port of the Character Mapper to the output port of the component that provides the inbound data. Link the output port of the Character Mapper to the input port of the component to which you want to direct the outbound data.

   You must configure the input port structure of the component to which to direct the outbound data.
3. Open the Character Mapper component window. If necessary, click the **Step to next incoming data buffer** icon on the toolbar to populate the input and output content.
4. Add a mapping definition.

See also

- *Creating New Mapping Definitions* on page 103

Creating New Mapping Definitions

Use the Character Mapper component window to define mapping rules for data that passes between the input port and output port.

1. Click the **Insert Mapping** icon on the toolbar, or right-click anywhere on the Mapping Definition pane, and select **Insert Mapping**.
2. Enter the character combination you want to replace in the From column, and the value with which you want to replace it in the To column.

   Character Mapper applies the rule, and displays the results in the Current Output Record pane.
   - To edit the record currently shown in the Current Input Record pane, click a row in the Current Input Port Content pane and make changes. The values in the Current Output Record pane and the selected row in the Current Outport Port pane are also updated.
   - To delete a mapping definition, right-click the mapping definition and select **Remove Mapping**.
   - To change the order of the mapping definitions, select the **Move row up and Move row down** icons on the toolbar.
3. Character Mapper applies the mapping, and updates the output results as soon as you write the rule. Click the **Enable auto refresh of output values** icon on the toolbar to toggle this kind of automatic synchronization.

   Note: Character Mapper applies mappings to all columns and all rows. To exclude columns from a character mapping, click **Exclude** in the Properties window, and select the columns you want to exclude.

Table 17. Mapping Notations

Type	Syntax	Example (@)
Character	%	@
ASCII decimal	<%>	<64>
ASCII hexadecimal	<0x%>	<0x40>
Unicode decimal	<u%>	<u0064>
Unicode hexadecimal	<u0x%>	<u0x0040>

Note: % represents the respective character code.

Table 18. Mapping Notations Allowing Special Characters To Be Mapped

Mapping	From	To
Umlaut internationalization	Ä	Ae
Keyword translation	kunde	customer
Delete CR LF from string	<13><10> or <0x0D><0x0A>	
Replace the lira currency symbol with the euro currency symbol	<u8356> or <u0x20A4>	<u8364> or <u0x20AC>

Note: The From and To values can contain any combination of characters in the specified notations.

Mapping Definitions Reuse

Export or Import character mapping definitions and save to a file, allowing them to be reused in other projects.

1. Export mapping definitions:
    1. Click the **Export character mapping** icon on the toolbar.
    2. Provide a file name and click **Save**. Character Mapper saves the file without an extension.
2. Import mapping definitions:
    1. Click the **Import character mapping** icon on the toolbar.
    2. Select the file you want to import and click **Open.**

Running Character Mapper Demos

Sybase IQ InfoPrimer includes a demonstration for the Character Mapper component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help > Demonstrations > Transform > Character Mapper**.
2. To access the sample projects, in the Navigator, select **Repository > TRANSFORMER.transformer.Repository > Projects**. Then select **Demo Character Mapper**.

Copy Splitter

The Copy Splitter component unconditionally copies input data to each output port. Unlike the Data Splitter, the Copy Splitter does not include any port expressions.

Using the Copy Splitter eliminates the need to invoke JavaScript, and reduces the cost of condition evaluation, thus improving performance.

By default, the Copy Splitter component has two output ports.

Configuring a Copy Splitter Component

Add and configure a Copy Splitter component.

1. Drag the **Copy Splitter** component onto the Design window.
2. Link the Copy Splitter input port to the output port of the component that provides the inbound data.
3. If required, create additional output ports.
4. Link the Copy Splitter output ports to the input ports of the components to which you want to direct outbound data.

Managing Copy Splitter Output Ports

Add and remove Output ports. Although the component is configured with two output ports, you can create additional output ports. New ports are identified by the name.

1. Add new output ports.
    1. Right-click the component and select **Add Output Port**.
    2. Enter a name for the new output port. Click **OK.**
2. Remove output ports.
    1. Right-click the port to remove and select **Remove Port**.

    If the output port is linked to another component, **Remove Port** is disabled. You must delete the link before deleting the port.

    Note: You cannot delete all ports. The component must have at least two output ports.

Data Calculator JavaScript

Data Calculator JavaScript is a Transformation component that lets you define rules that apply transformations to records that pass between input ports and output ports.

You can, for example, use Data Calculator JavaScript to define rules that transform port attributes or add rules that create new attributes.

Data Calculator can also perform lookups at the attribute level. You provide lookup data at special lookup ports.

Without lookups, Data Calculator JavaScript does not impact the simulation sequence. With lookups, all data is read into the lookup ports before the data at the main port is processed.

Configuring a Data Calculator JavaScript Component

Add and configure a Data Calculator JavaScript component.

1. Drag the Data Calculator JavaScript component onto the Design window.
2. If necessary, link the input port of Data Calculator to the output port of the component that provides the inbound data.
3. If the Data Calculator component window is open, click **Save** to close the window.
4. In the Design window, right-click the output port of Data Calculator, select **Assign Structure** and select one of these options:

Option	Action
IN	Create an output port structure that matches the input port structure of the Data Calculator.
Copy Structure	Open a window that allows you to apply an existing port structure to the output port.

5. In the Design window, double-click Data Calculator JavaScript, or, in the Properties window, click the **Rule** icon.
6. Select one of these options:

Options	Action
Yes	Create a set of transformation rules that maps each attribute in the IN-port to a corresponding attribute in the output port.
No	Manually map attributes in the input port to a corresponding attribute in the output port.

See also

- *Mapping Port Attributes* on page 107

Mapping Port Attributes

Although the Data Calculator component creates column-to-column mapping between the input port and output port as a default option, there may be times when you want to individually map port attributes.

1. On the Data Calculator component window, click the **Graph** tab.
2. Map the input port and output port in one of these ways:
   - Select **Mapping** in the menu bar, and select one of these predefined mapping sequences:
     - **Create Mapping by Order** – sequentially maps the port attributes of the input port and output port.

       Note: If the number of attributes is different, some port attributes are not mapped.
     - **Create Mapping by Name** – maps the port attributes of the input port and output port according to their names.
     - **Create Mapping by Name Case Sensitive** – maps the port attributes of the input port and output port according to their case-sensitive names.
     - **Create Mapping by Prefix** – maps the port attributes of the input port and output port by name, ignoring the specified prefixes.
     - **Create Mapping by Best Match** – maps the port attributes of the input port and output port that sound alike.
   - Connect the input port and output port attributes individually.

Displaying Transformation Results

The Data Calculator immediately displays the result of transformation rules, allowing you to verify incoming data, test transformation rules, and view the effect of rules on data output.

By default, output port values reflect IN-port values. Manually changing an input port attribute value affects only the data of the input port or output port buffer, allowing you to test transformation rules, and to see the results in the current output record. This is a convenient way to create test cases for transformation rules.

Use the Transformation Rule column to add, modify, or delete transformation rules. You can also edit single-line functions by changing the current attribute input field.

Note: The graphic view mirrors the actual port structures. You cannot add or delete attributes in the graphic view.

Managing transformation rules

- To add a transformation rule, right-click anywhere in the Transformation Rule or Current Output Port column, and select **Insert**. You can now use the added rules for further assignments or calculations.

- To delete a transformation rule, right-click a row in the Transformation Rule column, and select **Remove.**
- To change the order of the transformation rules, right-click the row in the Transformation Rule column, and select **Up** or **Down.**
- To add missing output attributes, click **Mapping**, and select **Add missing output attributes**.

Transformation rules are processed in sequential order. The processing starts with the first transformation rule of the list.

See also

- *JavaScript Editor and Debugger* on page 181

Data Calculator Simulation

Data Calculator lets you see changes applied as the data moves through the transformation rules. You may find this useful, for example, to see how changing a transformation rule affects outgoing data. Depending on the status of the Auto-Synchronization button, a transformation rule is immediately applied to the entire set of IN-port records after the rule is entered.

- Toggle auto synchronization – auto synchronization immediately applies all changes to the transformation rules to the current set of records at the input port.

  Note: If Auto-Synchronization is disabled, you can manually trigger the processing of the input port data by selecting the Step option.
- Manually apply all transformation rules to all current records at the input port. Click the **Step** icon on the toolbar.
- Fetch another set of records – click the **Step through the next incoming data buffer** icon on the toolbar.
- To step through input port records –
  - Click the appropriate record control on the toolbar.
  - Click **Navigate** and select the appropriate option.
  - Select a record from the **Input Port Content** list.
- Search for keywords in the transformation rules – click the **Search the Content of Transformation Rules** icon on the toolbar.

  Note: The values shown on the Current Input Record area are updated as you change the current record.
- Highlight null and empty values – click the **Highlight NULL-Values and Empty Values** icon on the toolbar.

Lookups in Data Calculator

Data Calculator performs lookups at the attribute level. You must enter lookup data at special lookup ports.

Adding Lookup Ports

Add lookup ports to the Data Calculator component.
Connect the output port of the data providing component directly to the Data Calculator component (not a port). A lookup port is automatically created and connected.

Alternatively, right-click **Data Calculator** and select **Add Input Port**. Connect the output port of the data providing component with the new port. You can add an unlimited number of lookup ports.

Preparing the Lookup Data

Prepare the lookup data.

Each lookup port must have at least two attributes. The first attribute represents the key. All other attributes represent return values.

To look up compound keys, concatenate the key values within a preceding component and use the same kind of concatenation on the key expression for the lookup.

Setting General Lookup Options

Set the lookup option for all lookup ports in the Properties window.
Use the Lookup Options property to configure the lookup. The Properties window displays a list of all lookup ports and current option values.

- Lookup Name – is inherited from the associated port and cannot be changed here. To change the name of a port, select **Description** from the port menu.
- Lookup Size – to optimize memory allocation and lookup performance, enter the estimated number of lookup records.
- Lookup Empty / Null – empty and null are normally handled as “unknown” keys, thus returning the lookup default value. If empty or null values are valid keys for your lookup, you can enforce looking up the values for these keys by activating this option.

Building Lookup Rules

Set up the lookup rules in the Data Calculator window.
Click the **Tabular** tab. If lookup ports are available, an additional Lookup column appears.

For each lookup rule, provide:

- Key Expression – is the value to search for in the first column of the lookup list. Enter the key expression (for example IN.PR_ID) as a transformation rule.
- Return Value – since lookup lists can have more than one return value column, you must specify which value to return, if the key is found. Select the associated port attribute from the pop-up menu on the Lookup column. For example: `LOOKUP1>>LOOKUP1.PR_NAME`.

  Note: Although return values are selected and displayed by name, the lookup internally uses the column number. This means you must review your lookup rules whenever the lookup port structure is modified, especially after you have added or removed attributes.

- Output Variable – the lookup return value is assigned to this variable. You can select a variable from the Output Port column (for example, `OUT>>OUT.PR_NAME`).
- (Optional) Default Expression – a lookup returns null if the given key value is not found. To return a different default value, enter an expression in the **Lookup Options** window. To open the Lookup Option window, click the icon in the Lookup column in the Transformation Rules and Current Output Record pane.

Running Data Calculator JavaScript Demos

Sybase IQ InfoPrimer includes several demonstrations for the Data Calculator component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Transform** > **Data Calculator - JavaScript**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer.Repository** > **Projects**.

   For demos without lookup, select:
   - **Demo Transfer U.S. Products**
   - **Demo Transfer German Products**
   - **Demo Data Calculator**

   For demos with lookups, select:
   - **Demo Transfer German Customers**
   - **Demo Transfer German Sales**
   - **Demo Transfer U.S. Customers**

Data Splitter JavaScript

The Data Splitter JavaScript component allows you to filter and distribute input data.

Configuring a Data Splitter JavaScript Component

Add and configure a Data Calculator JavaScript component.

1. Drag the **Data Splitter JavaScript** component onto the Design window.
2. Link the Data Splitter input port to the output port of the component that provides the inbound data.
3. Link the Data Splitter output ports to the input ports of the components to which you want to direct outbound data.
4. Select **Mutually Exclusive Rules** if you want port conditions to behave as if they are mutually exclusive. If this option is selected, ports are evaluated based on a user-specified order, instead of the order in which they appear on the component. When conditions are defined as mutually exclusive, the splitter writes a given record to 0 or 1 output port. The first output port that has a matching condition receives the input record. No subsequent port conditions are evaluated. If a condition does not match, the record is not written to any output port.

Note: The mutually exclusive port conditions are always evaluated in the order specified by the sequence number associated with each port condition.

5. Double-click the **Data Splitter JavaScript** component.
6. Add the conditions you want to use to direct the data flow:
    1. Double-click the port you want to add the conditions to. Alternatively, click the **Edit condition** icon on the port, or right-click the port and select **Edit condition**.
    2. On the Condition window, define the conditions for each column that you want to apply.
    3. Click **Save.**

Splitting Inbound Data

Adding the Data Splitter component to a project opens a component window that displays inbound data attributes and output port conditions.

The Data Splitter component is configured with two output ports. Both port conditions are preset to 1. Since this condition is always true, regardless of the current values of the input port, the Data Splitter copies all incoming records to both output ports.

The input port data buffer is initially empty, and only the inbound data attributes are visible. The output port structures match the input port structure.

To populate the input attributes, click the **Step to the next input buffer** icon on the toolbar. Input data appears in the upper part of the component window.

Since the output ports share the same port structure as the input port, selecting any record in the upper window causes the output port to indicate whether the record meets the port condition. When a record meets port conditions, the output port color is green. When a record does not meet port conditions, the output port color changes to red.

Inclusive Port Conditions

If the splitter mode is inclusive, that is, you have not defined mutually exclusive port conditions, each input record is tested against each port condition. For every matching port condition, a copy of the current record is written to the output port.

Exclusive Port Conditions

If you have defined mutually exclusive port conditions, ports are evaluated based on a user-specified order, instead of the order in which they appear on the component. The conditions are evaluated in order of first to last. The first output port with a matching condition receives the input record. No further port conditions are evaluated. You can change the evaluation order of conditions by clicking the **Edit evaluation order** icon and then swapping rows in the Evaluation order window. Click the **Reset evaluation order** icon to reset to the default order.

Note: The number of records that Data Splitter forwards to the output ports can differ from the number of incoming records. If the port condition is not defined as mutually exclusive, when a single record matches more than one port condition, it is available on all of these ports. Records that do not match any of the conditions are removed from the data stream. For mutually exclusive port conditions, a single input record can match only one output port. Records that do not match are not written to any output port.

Port Conditions Customization

You can assign a condition to each port. A condition consists of one or more expressions. Multiple expressions are concatenated by operators. When a condition is evaluated, the result is either true (1) or false (0).

Modifying Port Conditions

Edit an existing port condition.

1. Double-click the component. In the component window, right-click the port and select **Edit Condition**.
2. Create the conditions you want to apply to the port. You can:
   - Manually enter the conditions in the text area of the Condition window.
   - Drag and drop the variables and functions you want to add to your condition from the left pane to the text area. The Variables tab lists all the variables that you can use, and the Functions tab lists all available functions and operators that you can add to the condition.
   - Right-click the text area, and select the variables you want to add to the condition.

Adding New Output Ports

Although the component is configured with two output ports, you can create additional OUT-ports. New ports are identified by the name.

1. Click the **Add new port** icon on the port toolbar.
2. Enter a name for the new output port. Click **OK.**

Removing OUT-ports

Delete existing output ports.

1. Select the port to remove.
2. On the port toolbar, click the **Remove selected port** icon. Alternatively, right-click the port and select **Delete.**

Special Port Conditions

Port conditions determine how Data Splitter distributes records. In non-exclusive mode you can have multiple ports having some conditions.

- 1 – true. All records are forwarded to this port, including records that match any other port condition.
- (empty) – all records that do not match any other condition are forwarded to this port.

In exclusive mode you can have a single port having the condition 1– true. All records that do not match any preceding port condition are forwarded to this port.

Note: This condition must be the last in the evaluation order. Ports with empty conditions are invalid.

Running Data Splitter JavaScript Demos

Sybase IQ InfoPrimer includes several demonstrations for the Data Splitter component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Transform** > **Data Splitter - JavaScript**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer.Repository** > **Projects**. Then select:
   - Demo Data Splitter
   - Demo Text Data Sink Delimited/Fixed

   Note: Demos do not display the mutually exclusive port condition functionality.

SQL Executor

The SQL Executor component allows you to execute one or more custom SQL statements against a database server.

SQL Executor is a standalone component without input ports or output ports. You can place SQL Executor in a project separate from other components, or in a single project containing one or more SQL Executor components. For example, you can use SQL Executor to:

- Load data from a source table to a text file (with a format that IQ can support), if the source table allows extracting data into a file by using a SQL statement.
- Load data from a text file into the target IQ database, in a single transaction, using the **Load Table** command.

Configuring a SQL Executor Component

Add and configure a SQL Executor component to your project.

1. Drag the **SQL Executor** component onto the Design window.
2. In the Properties window, specify the properties for the component.

SQL Executor Properties List

The SQL Executor Properties list identifies the required and optional properties of the SQL Executor component.

Table 19. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.

Property	Description
Execute Script	Specify the custom SQL script to be executed.

Table 20. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Execute Success Script	Specify the custom SQL to be executed, if the Execute Script runs successfully.
Execute Error Script	Specify the custom SQL to be executed, if the Execute Script fails.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Lookup Components

A lookup operation looks up a value that corresponds to a key in a lookup table that contains a list of key and value pairs. A static lookup table can be cached during the execution of a project, or a lookup can be performed uncached and dynamically.

DB Lookup

The DB Lookup component looks up values in a database. The lookup data is specified by the result set of a query that returns exactly two columns—the lookup key and the lookup value.

You can assign the return value (lookup value) to any attribute of the current record. DB Lookup caches the lookup table during project execution. Changes that are applied to the underlying database during project execution have no effect on the lookup result.

Configuring a DB Lookup Component

Add and configure a DB Lookup component.

1. Drag the **DB Lookup** component onto the Design window.
2. In the Design window, connect the DB Lookups input port with the output port of the component that provides the inbound data.
3. In the Properties window, specify a valid interface and host name.
4. Specify the key attribute holding the value to look for and value attribute to take the lookup result value. You can replace a value by choosing the same attribute for both.
5. In the Properties window, click the **Query** icon to open the Query window.
6. On the Query window, create and save the query to retrieve the lookup data. Design your query to return the lookup key and the lookup value from the source table.

Example

Assume that you want to replace the product number used for German products by the product number used in the US. The German products are in the table `PRODUKTE(PR_NUMMER, PR_NAME, PR_PREIS)`. The IN-port of the DB Lookup component contains those three attributes.

The table to perform the lookup of the U.S. product number is `LOOKUP_PRODUCTS(SOURCE, DESTINATION)`. The `SOURCE` column contains the German product numbers and the `DESTINATION` column contains the U.S. product number.

If no value for the German `PR_NUMMER` can be found in the `LOOKUP_PRODUCTS`, the current `PR_NUMMER` is replaced by the string “INVALID”. A successful lookup replaces the German product number by the corresponding U.S. number.

To set up the DB Lookup Component for this example, select:

- Key Attribute – `PR_NUMMER`
- Value Attribute – `PR_NUMMER`

- Default Value – `INVALID`
- Query – `select SOURCE, DESTINATION FROM LOOKUP_PRODUCTS`

DB Lookup Properties List

The DB Lookup properties list identifies the connection parameters and other properties you define on the Database Configuration window.

Table 21. Required Properties

Property	Description
Key Attribute	Select a key attribute from the list of IN-port attributes. This attribute corresponds to the first column of the lookup table.
Value Attribute	Select the attribute to receive the value found by the lookup from the value attribute list. The lookup value returned overwrites any existing value. Both Key Attribute and Value Attribute may refer to the same attribute of the record structure, thus allowing a key to be overwritten with its corresponding value.
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Query	Create a query that retrieves information from the datasource. Use the Query window to create a simple query, or click the **Query Designer** icon to open the Query Designer.

Table 22. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Default Value	Assign a default value to the value attribute. DB Lookup uses this value when it cannot find the key value in the lookup table.
Use Key Value	Assign the key value to the value attribute instead of the default, if the lookup fails.
Lookup Empty/Null Keys	Perform a lookup for empty or NULL key values. Otherwise, the selected default method applies.
Lookup Size	Specify the estimated number of lookup records, to optimize memory allocation and lookup performance.

Property	Description
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
Database Options	Set options that override performance defaults and control the behavior of some transactions.

Running DB Lookup Demos

Sybase IQ InfoPrimer includes several demonstrations for the DB Lookup component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Lookup** > **DB Lookup**.
2. To access the sample projects, in the Navigator, select . Then select:**Repository** > **TRANSFORMER.transformer.Repository** > **Projects**
   - **Demo DB Lookup**
   - **Demo Transfer German Products**

DB Lookup Dynamic

The DB Lookup Dynamic component performs a dynamic lookup by referencing the key value in the query **WHERE** clause.

Unlike the DB Lookup component, DB Lookup Dynamic does not cache lookup information, and performs one SQL lookup for each record that passes the component. During project execution, the lookup table data may be modified by concurrent database users (or even within the same project). In this case, a database lookup that is not dynamic may search for invalid data. Use DB Lookup Dynamic to ensure you locate the current value.

Another typical use for this component is a lookup table that exceeds the memory available on the local machine. The DB Lookup Dynamic component provides a slower-performing lookup, but requires no cache memory, as it performs the lookup on a record-by-record basis.

Configuring a DB Lookup Dynamic Component

Add and configure a DB Lookup Dynamic component to your project.

1. Drag the **DB Lookup Dynamic** component onto the Design window.
2. In the Design window, connect the DB Lookups input port with the output port of the component that provides the inbound data.
3. In the Properties window, specify the interface and host name.
4. Specify the key attribute holding the value to look for and value attribute to take the lookup result value. You can replace a value by choosing the same attribute for both.
5. Specify a lookup key value to be used when designing and testing the lookup Query.
6. Click the **Query** icon to open the Query window.
7. On the Query window, create and save the query to retrieve the lookup data. Design your query to return the lookup value from the source table. Use the predefined variable Lookup in the **where** clause of your Query as a placeholder for the Lookup Key.

See also

- *SBN Expressions* on page 180

Resetting Default Lookup Key Values

During simulation, the Key Attribute values from the incoming data stream are assigned to the predefined variable Lookup.

To reassign the specified lookup key value to this variable:

1. Right-click the **DB Lookup Dynamic** component.
2. Select **Reset Lookup Key** value.

Example

Assume you want to replace the product number used for German products by the product number used in the US. The German products are in the table `PRODUKTE(PR_NUMMER, PR_NAME, PR_PREIS)`. Your IN-port for the DB Lookup Dynamic component therefore contains these three attributes.

The table to look up the U.S. product number is table `LOOKUP_PRODUCTS(SOURCE, DESTINATION)`. The `SOURCE` column contains the German product numbers and the `DESTINATION` column contains the US. product number.

If no value for the German `PR_NUMMER` can be found in the `LOOKUP_PRODUCTS`, the current `PR_NUMMER` is replaced by the string `“INVALID”`. A successful lookup replaces the German product number with the corresponding US. number.

To set up the DB Lookup Dynamic component for this example, select:

- Key Attribute – `PR_NUMMER`
- Value Attribute – `PR_NUMMER`
- Default Value – `INVALID`
- Query – `select DESTINATION FROM LOOKUP_PRODUCTS, where SOURCE = ‘[Lookup]’`

DB Lookup Dynamic Properties List

The DB Lookup Dynamic Properties list identifies the required and optional properties of the DB Lookup Dynamic component.

Table 23. Required Properties

Property	Description
Key Attribute	Select a key attribute from the list of input port attributes. This attribute populates the placeholder variable lookup.
Value Attribute	Select the attribute to receive the value found by the lookup from the value attribute list. The lookup value returned overwrites any existing value. Both key attribute and value attribute might refer to the same attribute of the record structure therefore allowing the overwriting of a key with its corresponding value.
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Query	Create a query that retrieves information from the datasource. Use the Query window to create a simple query, or click the **Query Designer** icon to open the Query Designer.

Table 24. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Default Value	Assign a default value to the value attribute, in case the key value was not found in the lookup table.
Use Key Value	Assign the key value to the value attribute instead of the default, if the lookup fails.
Lookup Empty/Null Keys	Perform the lookup even for empty or NULL key values. If not selected, the default method applies.

Property	Description
Lookup Key Value	Specify a value for the lookup key populating the lookup variable for testing the query at design time.
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period ("`.`") as the decimal separator.
Database Options	Set options that override performance defaults and control the behavior of some transactions.

Running DB Lookup Dynamic Demos

Sybase IQ InfoPrimer includes a demonstration for the DB Lookup Dynamic component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Lookup** > **DB Lookup – Dynamic**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer.Repository** > **Projects**, and then select **Demo DB Lookup Dynamic**.

Staging Components

Staging components have at least one input port and one output port, and apply specific transformations to the data in the transformation stream.

DB Staging

DB Staging loads incoming data streams into a single staging area. It buffers all incoming data, then creates an outgoing data stream, which represents the result set of a given **select** statement.

You can create staging tables based on the output port structure of the preceding component. Although many transformation components work on a record-by-record basis, the staging component works in two phases:

- Phase 1 – collect all records from the preceding components.
- Phase 2 – run the query and provide the records of the result set in blocks of a given size.

You can use staging components to perform sorts or aggregations by using **ORDER BY** or **GROUP BY** clauses in the Query property. You can join data from heterogeneous sources by loading them into multiple tables of the staging database. You can also use the DB Staging component to create an intermediate image of the transformation for further inspection or processing.

Note: In simulation, the DB Staging component first retrieves all data from the original datasources, then acts as a new datasource for subsequent components. The component allows the Read Block Size value of the original source components to be overwritten.

Configuring a DB Staging Component

Add and configure a DB Staging component.

1. Drag the **DB Staging** component onto the Design window.
2. Connect the DB Staging input port to the component that provides the inbound data.

   To add input streams to the DB Staging component, you can drop connections from the data providing component on the staging component. The ports are automatically created by the component.
3. In the Properties window, add the Connection Parameters to the database where you want to add staging tables.

   Specify a valid interface and host name.

   Note: If the staging tables you are going to use already exist, skip the next step.
4. In the Design window, right-click the DB Staging component and select one of:
   - **Create Staging Table from Input** – select the appropriate port structure, and click **OK**. Enter a name for the new table.
   - **Create Staging Table from Port** – enter a name for the new table and select an appropriate port structure. Click **Apply.**
5. In the Add table window, verify that the new table information is correct and click **Create.**
6. In the Properties window, click the **Stage Options** icon.

   The Stage Options window lets you define the Truncate Table and Write Block Size options for each staging table.
7. In the Properties window, click the **Query** icon and create a query to select data from the staging area.

DB Staging Properties List

The DB Staging Properties list identifies the required and optional properties of the Data Staging component.

Table 25. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Stage Options	Set the staging options.
Query	Create a query that retrieves information from the datasource. Use the Query window to create a simple query, or click the **Query Designer** icon to open the Query Designer.

Table 26. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Read Block Size	Determine the number of records retrieved by the component in a single step.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
PreOutput Processing SQL	Specify an additional SQL script that is executed after all the data from the transformation flow has been loaded into the tables associated with the input ports, and before the query resultset is retrieved. This property enables you to modify or update the staging tables before the output is created.
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.

Property	Description
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.

Property	Description
Load Stage Path	Specify a data file path. The load stage file must reside on the same machine as the IQ Server. When using a Sybase IQ database, if you specify a Load Stage Path, the component uses the **LOAD TABLE** statement instead of using SQL statements. This leads to faster performance. **Note:** You need not specify the Load Stage Path when client-side loading is enabled for your Sybase IQ 15.0 and later staging database with ODBC interface. The IQ server automatically uses its default **LOAD TABLE** statement to add records from files located on the remote host machines into the Sybase IQ table. To create a pipe, specify pipe:// as the Load Stage Path parameter. A pipe is not used if the Load Stage Path is blank. If you use named pipes on UNIX or Linux, the Sybase IQ InfoPrimer Server and the IQ Server must reside on the same machine. This is not a requirement for Windows.
Load Stage (Server)	Specify the server path to the data file or, leave it empty when using a pipe. If the Sybase IQ server must use a different path to the temporary data file than specified in the Load Stage property, enter it here. **Note:** You need not specify this property if the file is on the same machine as the grid engine and client-side loading is enabled for your Sybase IQ 15.0 staging database with ODBC interface. If client-side loading is enabled, the IQ server automatically uses its default **LOAD TABLE** statement to add records from files located on the remote host machines into the Sybase IQ table.
Create Tables	Specify if you want to automatically create tables based on the input port structures at runtime. Select: • None (Default) – to not automatically create tables. If a specified table does not exist, an error is thrown. • Non-existing – to create tables that do not exist, based on the structure of the associated input port. • All – to drop and re-create all tables associated with input ports based on the port structure. **Note:** This property applies to all tables associated with input ports. It cannot be specified at the table level.

Property	Description
Drop Tables	Specify to remove the tables that are created at runtime, after the project finishes processing. This property applies to all tables associated with input ports. It cannot be specified at the table level.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.
Database Options	Set options that override performance defaults and control the behavior of some transactions.

Running DB Staging Demos

Sybase IQ InfoPrimer includes a demonstration for the DB Staging component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help > Demonstrations > Staging > DB Staging**.
2. To access the sample projects, in the Navigator, select **Repository > TRANSFORMER.transformer.Repository > Projects** and then select Demo DB Staging.

Destination Components

Destination components (also called data sinks) write data to specific targets. This component type has one input port and no output port.

Preconditions for Using DB Data Sink Components for Bulk Loading

You must enter the Load Stage Path for the DB Data Sink components to support IQ bulk loading. To support IQ bulk loading with client-side loading, the DB Data Sink components must meet some conditions.

- Sybase IQ version must be 15.0 or above.
- The **allow_read_client_file** option must be on.
- Interface must be ODBC.
- Do not specify the Load Stage Path.

Additionally:

- The **Insert** options should not have a **SQL Insert** value specified in case of the DB Data Sink Insert component.

- The **Update** options should not contain a **SQL UPDATE SET** clause, and none of the NON-NULLABLE column should be omitted from the list of update columns, in case of the DB Data Sink Update component.

See also

- *Enabling Client-Side Load Support* on page 127

DB Bulk Load Sybase IQ

DB Bulk Load Sybase IQ is a destination component that performs bulk operation on a Sybase IQ table.

Use this component to manipulate records of a table in a Sybase IQ database based on the records from the IN-port of the component.

Configuring a DB Bulk Load Sybase IQ Component

Add and configure a DB Bulk Load Sybase IQ component to your project.

1. Drag a **DB Bulk Load Sybase IQ** component onto the Design window.
2. Connect the input port of the DB Bulk Load Sybase IQ component to the output port of the component that provides the inbound data.
3. In the Database Configuration window, add the Sybase IQ connection parameters.
4. Click the **Destination** icon and select the table where you want to load the inbound data. To write to a new destination table, see Adding new Sybase IQ destination tables.
5. Click the **Load Stage** icon. You can either:
   - Select or enter the file name you want to use as a temporary data file and click **Save**, or,
   - Add a pipe name in the **Load Stage** field (syntax: `pipe://`).
6. Click **Finish.**

To add DB Bulk Load IQ to your project:

- Sybase IQ must be up and running before you add the component to your project. You can increase performance if you run both Sybase IQ INfoPrimer Server and the IQ database on the same machine; however, this is not required.
- To connect to the target database on IQ, select a valid host name and interface.
- To load the data into a new IQ table, you can create a destination table based on DB Bulk Load's IN-port or the structure of any available port in the project.
- To customize the script, right-click the DB Bulk Load IQ component, and select **Generate Load Script**. Click the **Load Script** icon in the Properties window, edit, and save your script.

See also

- *Adding a Destination Table* on page 127

- *DB Bulk Load Sybase IQ Properties List* on page 132

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.
Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

Adding a Destination Table

You can write the transformation results or inbound data to an existing table or add a destination table based on existing ports in the project. You cannot add a table based on a port structure from the Database Configuration window or Properties window. You must select the port structure in the Design window.

Writing to an Existing Table

Write the transformation results to a selected table that already exists.

1. In the Database Configuration window, specify the table where you want to write the transformation results. You can click the **Destination Table** icon to select the table, or manually enter the name in the `Destination Table` field.
2. Click **Finish.**

Adding a Destination Table Based on Input Port

Create a new destination table based on input ports for writing inbound data.

1. Right-click the component and select **Add Destination Table from Input**.
2. Name the new table. Click **OK.**
3. Verify that the table information is correct, and click **Create.**

Adding a Destination Table Based on an Existing Port

Create a new destination table based on existing port for writing inbound data.

1. In the Design window, right-click the component, and select **Add Destination Table from Port**.
2. Name the new table. Click **OK.**
3. Select the port structure to assign to the new table. Click **Apply.**
4. In the Add table window, verify if the table information is correct and click **Create.**

Enabling Client-Side Load Support

You can load data into the Sybase IQ table, from files that are located on a different host machine than Sybase IQ. You need not install Sybase IQ InfoPrimer Development and Sybase

IQ on the same machine; Sybase IQ InfoPrimer Server and Sybase IQ can communicate in a networked environment, allowing you to bulk-load from a remote machine in a single step.

To support client-side load support:

- Install the Sybase IQ 15.0 client on the same machine as Sybase IQ InfoPrimer Server.
- Install the Sybase SQL Anywhere 11 ODBC driver on the same machine as Sybase IQ InfoPrimer Development and Sybase IQ InfoPrimer Server.
- The target IQ database version must be Sybase IQ 15.x.
- On each Sybase IQ 15.x server, enable the **allow_read_client_file** and **allow_write_client_file** options. To set these options:
  1. From Sybase Central, connect to the Sybase IQ 15.x server.
  2. Right-click the Sybase IQ server database name and select **Options**.
  3. Select **allow_read_client_file** and **allow_write_client_file** options and change their values to on. By default, the value is off.
  4. Enable the **allow_read_client_file** server option property using the **isql** or **dbisql** utility:

     ```
     set option allow_read_client_file=on
     GRANT READCLIENTFILE TO <group | user>
     ```

Once you have completed these prerequisites, select **Use IQ Client Side Load** in the Properties window of the component. Also, select ODBC as the interface, or you may encounter errors loading data from remote host machines. Client-side loading works only with ODBC when the ODBC driver being used is an IQ 15.0 ODBC driver.

Note: If you select **Use IQ Client Side Load** to enable bulk loading of data into the IQ database from files located on client machines, provide a file path name instead of a pipe name in the Load Stage property field. Client-side loading is not supported using Load Stage pipe names.

You are required to make some additional configurations to enable support for multiplex execution by using multiple writers for loading data to IQ.

Multiple Writers Configuration for Loading Data

Sybase IQ InfoPrimer supports the multiple writer functionality that is available in Sybase IQ 15.0; you can add multiple writers for loading data into an IQ database. Multiple writers allow parallel loading of Sybase IQ tables, which is faster than sequential loading.

You can use multiple writers if you have:

- Selected more than one table from the source database and you are migrating to more than one table in the target IQ database.
- Created a job with a multiproject component involving multiple tables, or if you have selected multiple projects that are linked to parallel execution topology for job execution.

To use the multiple writer functionality, you must have these permissions in the target IQ database:

Object Name	Type	Required Permission
ETL_MULTIPLEX_STATE	Table	**create**
ETL_MULTIPLEX_VERSION	Table	**create**
sp_iqstatistic	Stored Procedure	**execute**

Note: Sybase IQ InfoPrimer creates two tables, `ETL_MULTIPLEX_STATE` and `ETL_MULTIPLEX_VERSION` in the IQ database. Each row in the `ETL_MULTIPLEX_STATE` table signifies an IQ writer selected by an Sybase IQ InfoPrimer grid node, which is automatically removed after each execution. If a grid node fails due to an unexpected error, you must manually clean the data in this table.

You can set the required permissions using Sybase Central:

1. In Sybase Central, connect to the Sybase IQ 15.0 server.
2. Expand Users & Groups, then select the user for whom you want to set the create table permission.
3. Right-click the user and select **Properties**.
4. Select the **Authorities** tab and click the **Resource** option to give the user permission to create database objects in the IQ database.
5. Select the **Permissions** tab and then select the **Procedures & Functions** option to see a list of available permissions.
6. Select **sp_iqstatistics** and click the corresponding **Execute** column to give the user permission to execute the stored procedure in the IQ database.
7. Click **OK** to save the settings.

Note: To support multiplex execution, you must install the SQL Anywhere 11 ODBC driver on the same machine as Sybase IQ InfoPrimer Development and Sybase IQ InfoPrimer Server.

Defining Preferred Writers in the IQMultiplex.ini File

Update the `IQMultiplex.ini` file to define preferred writers for multiplex execution.

1. Navigate to the `etc` directory in the installation folder and use a text editor to open the `IQMultiplex.ini` file.
2. Add one section for each multiplex group you want to use. By default, the `IQMultiplex.ini` file is empty. If you do not specify anything, the grid engine uses the internal default values.

   A sample section is as follows:

   ```
   [dbsybase15+iq15m+sample]
   /* This is the section name */
   Enabled=true
   Workload=OperationsWaiting
   MinimalUpdateInterval=60
   ```

```
SelectWriterTimeout=6
MostIdleNode=select NAME from TEST_CUSTOMER_NODE
order by WORKLOAD asc
MostBusyNode=select NAME from TEST_CUSTOMER_NODE
order by WORKLOAD desc
```

You must provide the section name, which includes the coordinator's interface name, host name, and database name, separated by a plus (+) sign. Do not use colon (:), hash (#), or equal sign (=) characters.

3. Save and close the file.

Table 27. Multiplex Group Optional Properties

Name	Type	Value	Description
Enabled	boolean	True (default) or False	Enable or disable this feature in the IQ server database.
Workload	text	OperationsWaiting (default)	Specify which row of the result of **EXEC sp_iqstatistics** should be used to take as workload.The dispatcher executes the stored procedure **EXEC sp_iqstatistics** to query the writer workload. The query result set returns the operation status name in the second column, and the status value in the fourth column. The dispatcher finds the row for which the second column matches the value of the workload option you specify, and then uses the fourth column of that row as the final workload value.
MinimalUpdateInterval	integer	Greater than zero (0), default value is 6	The minimal interval, in seconds, for the dispatcher to refresh writer information by querying the coordinator.
SelectWriterTimeout	integer	Greater than equal to zero (0), default value is 0	The number of seconds the dispatcher should wait, if all the writers are selected and not released. When you specify 0, the dispatcher waits indefinitely. In case of a timeout, an error is generated.

Name	Type	Value	Description
MostIdleNode	SQL	Empty by default	The first column of the first row returned by the SQL execution should be a writer node name. The dispatcher assumes the returned writer as the most idle node in the multiplex, and uses it as the target for the next table load request. For example, this SQL script creates a custom dispatch table: **DROP TABLE TEST_CUSTOMER_NODE; CREATE TABLE TEST_CUSTOMER_NODE(NAME varchar(100), WORKLOAD int /*must be integer*/); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w1', 78); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w2',34); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w3',12); /* iq15w1-w3 are writers*/** To obtain the most idle node from the table, add this SQL query in the `IQMultiplex.ini` file: **Select NAME from TEST_CUSTOMER_NODE order by WORKLOAD asc** iq15w3 is returned as the most idle node.

Name	Type	Value	Description
MostBusyNode	SQL	Empty by default	The first column of first row returned by the SQL execution should be a writer node name. The dispatcher assumes the returned writer as the busiest node in the multiplex, and delays using it as a load table target. For example, this SQL script creates a custom dispatch table: **DROP TABLE TEST_CUSTOMER_NODE; CREATE TABLE TEST_CUSTOMER_NODE(NAME varchar(100), WORKLOAD int /*must be integer*/); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w1', 78); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w2',34); INSERT INTO TEST_CUSTOMER_NODE (NAME,WORKLOAD) VALUES('iq15w3',12); /* iq15w1-w3 are writers*/** To obtain the most busy node from the custom dispatch table, add this SQL query in the `IQMultiplex.ini` file: **Select NAME from TEST_CUSTOMER_NODE order by WORKLOAD desc** iq15w1 is returned as the most busy node.

DB Bulk Load Sybase IQ Properties List

DB Bulk Load Sybase IQ properties list identifies the connection parameters and other items you define on the Database Configuration window.

Table 28. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Destination	Select the destination table from a set of existing tables.

Property	Description
Load Stage	Specify a data file path or pipe name. The load stage file must reside on the same machine as the IQ Server. If you use named pipes on UNIX or Linux, the Sybase IQ Info-Primer Server and the IQ Server must reside on the same machine. This is not a requirement for Windows. **Note:** If you have selected the **Use IQ Client Side Load** option, provide a file path name instead of a pipe name in the Load Stage field. Client-side loading is not supported using named pipes.

Table 29. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Shared Connection	Allow the component to share a single connection to the database with other target components that have identical connection and database parameters. A connection can be shared only between components within the same project. Components in different projects within the same job cannot share a connection. Components that use the same database interface and login information, but use different database options cannot share a connection, and generate an error when the project is executed or simulated. **Note:** Connection sharing is not supported if the Use IQ Multiplex property is enabled.
Key	Select target key attributes to identify the records on Upsert or Delete operations. If no key is selected, the interface works with the primary-key information, which is delivered from the DB host. An error appears if no primary key information is available.

Property	Description
Function	Select one of these load functions: • Insert (default) – load records directly into the selected target table using the specified file path or pipe name. • Upsert – update existing records and insert the new records. Existing records are replaced and not updated on an attribute level. You can use the Key property to specify the target attributes identifying the records you want to update. • Delete – delete records from the target tables based on the keys in the incoming data. You can use the Key property to specify the target attributes identifying the records you want to delete. If you have selected the Truncate option, all records are removed from the target table before loading. In this scenario, the selected functions perform as follows: • Insert and Upsert load all records to the target table directly. • Delete does not move any records, but preprocessing and postprocessing SQL statements are still executed.
Truncate	Remove all records from the destination table before the load.
Use IQ Client Side Load	Add records from files located on remote host machines into the Sybase IQ table, using the **LOAD TABLE** statement.
Load Script	The **LOAD TABLE** statement is generated at runtime based on the component settings, if this property is empty. To use a customized script, right-click the component and select **Generate Load Script**. The **LOAD TABLE** script is generated for Insert. After you generate the script, you can click **Load Script** and edit the script. **Note:** If a custom load script is provided, the Function property is ignored.
Load Stage (Server)	Specify the server path to the data file or, leave it empty when using a pipe. If the Sybase IQ server needs to use a different path to the temporary data file than specified in the Load Stage property, enter it here.

Property	Description
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;). **Note:** If you have selected the Truncate option, all records from the destination table are deleted before the preprocessing SQL statement is executed.
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.

Property	Description
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.
Use IQ Multiplex	Support multiplex execution by using multiple writers to load data into the IQ database.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

DB Bulk Load Sybase IQ and DB Space

If you are using the Bulk Load Sybase IQ component and the project or job takes a long time to execute, and if you see an `out of space` message in the Sybase IQ console or log, you need to add another dbspace.

The message in the IQ message file indicates which dbspace has run out of space and the minimum number of megabytes to add. If the problem occurs while you are inserting data, you may need more room in the IQ store. If the problem occurs during queries with large sorts and merges, you may need more room in the temporary store.

This SQL statement adds 100MB of database space to the existing `iqdemo` database on Windows:

```
CREATE DBSPACE iqdemo2 AS
'd:\\sybase\\IQ-15_3\\demo\\iqdemo2.iq'
IQ STORE
SIZE 100;
```

This SQL statement adds 200MB of temporary space to the existing `iqdemo` database:

```
CREATE DBSPACE iqdemotmp AS
'd:\\sybase\\IQ-15_3\\demo\\iqdemo2.iqtmp'
IQ TEMPORARY STORE
SIZE 200 ;
```

Note: For additional information about diagnosing potential memory problems, see *Sybase IQ Troubleshooting and Recovery Guide > Resource issues*.

Customizing the IQ Loader Data Format

The IQ Loader Interface uses default values for delimiters, null handling, and character sets when writing to the data file or pipe, and when generating the LOAD TABLE script.

Table 30. Default Values in the Sybase IQ InfoPrimer Server `INI` File

Group	Key	Values	Default	Description
iq_loader	rowdelim	Any string. '\n' for line break.	' \n '	Line delimiter
iq_loader	coldelim	Any string. '\t' for tab.	' |@#& '	Column delimiter
iq_loader	nullre-place	Any string. If empty, NULL clause is not added to the load script.	' [NULL] '	String used for NULL val-ues
iq_loader	character-set	Any character set supported by IQ.	' ' (=auto)	Encoding used in data file

DB Data Sink Delete

DB Data Sink Delete removes records from a database destination table that match the incoming values of a selected key. If there are no matching records, DB Data Sink Delete does not display an error message.

Configuring a DB Data Sink Delete Component

Add and configure a DB Data Sink Delete component to your project.

1. Drag the **DB Data Sink Delete** component onto the Design window.
2. In the Database Configuration window, add the connection parameters for the target database. Specify a valid interface and host name.
3. Specify the table to write the transformation results. You can click the **Destination Table** icon to select the table, or manually enter the name in the `Destination Table` field.

   You can write the transformation results to an existing table or add a destination table based on existing ports in the project.

   To add a table based on an existing port structure, skip this step.
4. Click Finish.
5. In the Properties window, click the Key icon to select the columns identifying the records to remove from the Destination table.

   You must select a Destination table before you can select a key, and you can select multiple key columns. This is a logical selection, not related to any underlying indexes in the database schema.
6. Specify any other optional properties in the Properties window.

See also

- *DB Data Sink Delete Properties List* on page 138
- *Adding a Destination Table* on page 127

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.
Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

Loading Data at Input Port to a Database or a Text File

Use the Flush Buffer option to write out the buffered rows to the target. All components for which a write block size has been specified can buffer data until the write block size is reached. At any time during simulation, if there is data in the buffer, the Flush Buffer option appears along with the number of rows that are yet to be written to the target. If the buffer is empty, the option is dimmed.
Right-click the **DB Data Sink Delete** component and select **Flush Buffer**.

DB Data Sink Delete Properties List

The DB Data Sink Delete Properties list identifies the required and optional properties of the DB Data Sink Delete component.

Table 31. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Destination Table	Select the destination table. You can also create a new destination table based on a components port structure.
Key	Select the columns of the destination table that identify the records to delete. You must select a destination table before you can select a key. You can select multiple key columns.

Table 32. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Shared Connection	Allow the component to share a single connection to the database with other target components that have identical connection and database parameters. A connection can be shared only between components within the same project. Components in different projects within the same job cannot share a connection Components that use the same database interface and login information, but use different database options cannot share a connection, and generate an error when the project is executed or simulated.
Write Block Size	Specify the number of records to be written to the file in a single write operation.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Opening Attribute Quote	Prefix for attribute names in SQL statements.
Closing Attribute Quote	Postfix for attribute names in SQL statements.
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.

Property	Description
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.
Load Stage Path	Specify a data file path. The load stage file must reside on the same machine as the IQ Server. When using a Sybase IQ database, if you specify a load stage path, the component uses the **LOAD TABLE** statement instead of using SQL statements. This leads to faster performance. **Note:** You do not have to enter the load stage path if client-side load-balancing capability is available on your IQ Server. It is automatically used with the **LOAD TABLE** statement to add records from files located on remote host machines into the Sybase IQ table. To create a pipe, specify pipe:// as the load stage parameter. A pipe is not used if the load stage is blank. If you use named pipes on UNIX or Linux, the Sybase IQ InfoPrimer Server and the IQ Server must reside on the same machine. This is not a requirement for Windows.

Property	Description
Load Stage (Server)	Specify the server path to the data file or, leave it empty when using a pipe. If the Sybase IQ server must use a different path to the temporary data file than specified in the Load Stage property, enter it here.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Running the DB Data Sink Delete Demos

Sybase IQ InfoPrimer includes several demonstrations for the DB Data Sink Delete component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Destination** > **DB Data Sink - Delete**.
2. To access the sample projects, in the Navigator, select **Repository** >**TRANSFORMER.transformer.Repository** > **Projects**, and then select **Demo DB Datasink Delete**.

DB Data Sink Insert

DB Data Sink Insert adds records from the IN-port to a database table. You can exclude attributes or assign default values to determine the records you insert into the table.

Configuring a DB Data Sink Insert Component

Add and configure a DB Data Sink Insert component to your project.

1. Drag the **DB Data Sink Insert** component onto the Design window.
2. In the Database Configuration window, add the connection parameters for the target database.
3. Specify the appropriate properties.
4. Select or enter the destination table.

   You can write to an existing table or add a table based on existing ports in the project. To add a table based on an existing port structure, skip this step.
5. Click **Finish.**

See also

- *DB Data Sink Insert Properties List* on page 142

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.

Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

Loading Data at Input Port to a Database or a Text File

Use the Flush Buffer option to write out the buffered rows to the target.

All components for which a write block size has been specified can buffer data until the write block size is reached. At any time during simulation, if there is data in the buffer, the Flush Buffer option appears, along with the number of rows that are yet to be written to the target. If the buffer is empty, the option is not available.

Right-click the **DB Data Sink Insert** component and select **Flush Buffer**.

DB Data Sink Insert Properties List

The DB Data Sink Insert Properties list identifies the required and optional properties of the DB Data Sink Insert component.

Table 33. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Destination Table	Select the destination table from a set of existing tables, or enter the table manually. You can also create a new destination table based on a components port structure.

Table 34. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.

Property	Description
Shared Connection	Allow the component to share a single connection to the database with other target components that have identical connection and database parameters. A connection can be shared only between components within the same project. Components in different projects within the same job cannot share a connection. Components that use the same database interface and login information, but use different database options cannot share a connection, and generate an error when the project is executed or simulated.
Insert Options	Determine how records will be inserted. The Include columns specify the attributes to get values assigned from the component. Unselect any attributes for which you want to apply the database defaults. In the **SQL INSERT** clause column, you can overwrite the value of the incoming attribute with a new one. You can use any expression allowed in the SQL language of the underlying database. SBN expressions are evaluated during initialization of the component, so the value or expression is always constant during a single execution. For example, SQL INSERT value clauses are: • Static value – ‘valid’ • Dynamic value (evaluated by Sybase IQ InfoPrimer Server) – ‘[**uDate(“now”)**]’ • Database function (evaluated by database server) – **getdate**()
Truncate Table	Remove all records from the destination table when initializing the transformation process.
Write Block Size	Specify the number of records to be written to the file or pipe in a single write operation.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Opening Attribute Quote	Specify a prefix for attribute names in SQL statements.
Closing Attribute Quote	Specify a postfix for attribute names in SQL statements.

Property	Description
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.

Property	Description
Load Stage Path	Specify a data file path. The load stage file must reside on the same machine as the IQ Server. When using a Sybase IQ database, if you specify a load stage path, the component uses the **LOAD TABLE** statement instead of using SQL statements. This leads to faster performance. **Note:** You do not have to enter the load stage path if client-side load-balancing capability is available on your IQ Server. If available, it is automatically used with the **LOAD TABLE** statement to add records from files located on remote host machines into the Sybase IQ table. To create a pipe, specify pipe:// as the Load Stage parameter. A pipe is not used if the Load Stage is blank. If you use named pipes on UNIX or Linux, the Sybase IQ InfoPrimer Server and the IQ Server must reside on the same machine. This is not a requirement for Windows.
Load Stage (Server)	Specify the server path to the data file or, leave it empty when using a pipe. If the Sybase IQ server needs to use a different path to the temporary data file than specified in the Load Stage property, enter it here.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Running the DB Data Sink Insert Demos

Sybase IQ InfoPrimer includes several demonstrations for the DB Data Sink Insert component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Destination** > **DB Data Sink - Insert**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer. Repository** > **Projects**. Then select:
    - **Demo Transfer German Customers**
    - **Demo Transfer German Products**
    - **Demo Transfer German Sales**

- **Demo Transfer U.S. Customers**
- **Demo Transfer U.S. Products**

DB Data Sink Update

DB Data Sink Update component updates or overwrites all records that match a selected key. This component does not insert new records. If there are no matching records, DB Data Sink Update does not display an error message.

Note: If the update values violate any restrictions of the underlying table or object, such as constraints, referential integrity, or unique index definition, you see an error message. The selected key value attribute is independent of any existing index definitions.

Configuring a DB Data Sink Update Component

Add and configure a DB Data Sink Update component to your project.

1. Drag the **DB Data Sink Update** component onto the Design window.
2. Add the connection parameters for the target database.
3. Specify the destination table to which you want to write the transformation results. You can click the **Destination Table** icon to select the table, or manually enter the name of the table in the `Destination Table` field.

   You can write the transformation results to an existing table or add a Destination table based on existing ports in the project.

   To add a table based on an existing port structure, skip this step.
4. Click **Finish.**
5. In the Properties window, click the **Key** icon to select the columns of the Destination table that identify the records to update.

   You must specify a destination table before you can select a key, and you can select multiple key columns. This is a logical selection, not related to any underlying indexes in the database schema.

See also

- *DB Data Sink Update Properties List* on page 147

Updating Port Structure with Database Changes

Apply database changes to the port structure of components.
Right-click the component and select **Reconfigure.**

The Reconfigure option updates the component configuration when there is a change in the database schema. It closes the current connection, opens a new connection to the database, reads the metadata of the query, and applies the updates to the port structure.

Loading Data at IN-port to a Database or a Text File

Use the Flush Buffer option to write out the buffered rows to the target.

All components for which a write block size has been specified can buffer data until the write block size is reached. At any time during simulation, if there is data in the buffer, the Flush Buffer option appears, along with the number of rows that are yet to be written to the target. If the buffer is empty, the option is not available.
Right-click the **DB Data Sink Update** component and select **Flush Buffer**.

DB Data Sink Update Properties List

The DB Data Sink Update Properties list identifies the required and optional properties of the DB Data Sink Update component.

Table 35. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.
Destination Table	Select the destination table from a set of existing tables, or enter the destination table manually. You can also create a new destination table based on a components port structure.
Key	Select the columns of the destination table that identify the records to update. You must select a destination table before you can select a key, and you can select multiple key columns.

Table 36. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.

Property	Description
Shared Connection	Select this option to allow the component to share a single connection to the database with other target components that have identical connection and database parameters. A connection can be shared only between components within the same project. Components in different projects within the same job cannot share a connection. Components that use the same database interface and login information, but have differing database options cannot share a connection, and generate an error when the project is executed or simulated.
Update Options	Select the attributes (key attributes are not listed) to include in the update. By default, all attributes are selected. Unselect the attributes to exclude from the update. In the **SQL UPDATE SET** clause column, you can overwrite the value of the incoming attribute. In SQL language notation the, contents of the columns are processed as: `UPDATE customersSET cu_createdate = '2005-02-26'WHERE ….` You can use any expression allowed in the SQL language of the underlying database. SBN expressions are evaluated during initialization of the component, so the value or expression is always constant during a single execution.
Write Block Size	Specify the number of records to be written to the file or pipe in a single write operation.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Opening Attribute Quote	Specify a prefix for attribute names in SQL statements.
Closing Attribute Quote	Specify a postfix for attribute names in SQL statements.

Property	Description
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Standardize Data Format	Convert incoming `DATE` and `NUMBER` information into a standard format that Sybase IQ InfoPrimer can move between systems that support different formats. Dates are converted into this format: `YYYY-MM-DD hh:mm:ss.s`. For example: `2005-12-01 16:40:59.123` Numbers are converted using a period (“`.`”) as the decimal separator.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.

Property	Description
Load Stage Path	Specify a data file path. The load stage file must reside on the same machine as the IQ Server. When using a Sybase IQ database, if you specify a load stage path, the component uses the **LOAD TABLE** statement instead of using SQL statements. This leads to faster performance. **Note:** You do not have to enter the load stage path if client-side load-balancing capability is available on your IQ Server. If available, it is automatically used with the **LOAD TABLE** statement to add records from files located on remote host machines into the Sybase IQ table. To create a pipe, specify pipe:// as the Load Stage parameter. A pipe is not used if the Load Stage is blank. If you use named pipes on UNIX or Linux, the Sybase IQ InfoPrimer Server and the IQ Server must reside on the same machine. This is not a requirement for Windows.
Load Stage (Server)	Specify the server path to the data file or, leave it empty when using a pipe. If the Sybase IQ server needs to use a different path to the temporary data file than specified in the Load Stage property, enter it here.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Running the DB Data Sink Update Demos

Sybase IQ InfoPrimer includes a demonstration for the DB Data Sink Update component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Demonstrations** > **Destination** > **DB Data Sink - Update**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer. Repository** > **Projects**, and then select **Demo DB Datasink Update**.

Text Data Sink

Text Data Sink is a Destination component that writes transformation results to a text file in a delimited or fixed-length format.

Configuring a Text Data Sink Component

Add and configure a Text Data Sink component.

Note: Text Data Sink does not impact the simulation sequence.

1. Drag the **Text Data Sink** component onto the Design window.

   The output port of Text Data Sink should link to the input port of the component that provides the inbound data when you add the component to the project.
2. Click **Save.**

Note: If there is an adjacent component available in the Design window, Sybase IQ InfoPrimer automatically creates a link between the Text Data Sink input port with the output port of that component. For delimited files, this link provides the initial port structure you see when the window opens.

If not, you may need to close the Design window and connect the Text Data Sink input port with the output port of an adjacent component.

See also

- *Text Data Sink Properties List* on page 153
- *Modifying the Port Structure (Delimited Files)* on page 152
- *Exporting File Definitions* on page 152
- *Importing File Definitions* on page 152

Loading Data at Input Port to a Database or a Text File

Use the Flush Buffer option to write out the buffered rows to the target.

All components for which a write block size has been specified can buffer data until the write block size is reached. At any time during simulation, if there is data in the buffer, the Flush Buffer option appears, along with the number of rows. If the buffer is empty, the option is not available.
Right-click the **Text Data Sink** component and select **Flush Buffer**.

Exporting File Definitions

Export and Import options on the Component window let you save file properties to a definition file, and reuse them for other components. Export saves the component properties to a definition file.

1. To open the Component window, double-click Text Data Sink.
2. Click **Properties** > **Export**.
3. Select the definition file you want to use.

Importing File Definitions

Import loads a definition file you created with the Export command.

1. To open the Component window, double-click Text Data Sink.
2. Click **Properties** > **Import**.
3. Select the definition file you want to use.

Modifying the Port Structure (Delimited Files)

Column values on the Text Data Sink Components window reflect the current IN-port structure. You can assign a new port structure or re-create the current port structure.

Assigning a New Port Structure

Assign a new port structure using the component window.

1. To open the Component window, double-click Text Data Sink.
2. In the Column Names pane, click the Assign Port Structure icon.
3. Select the port structure to assign.

Regenerating Column Definitions

Regenerate column definitions using the component window.

1. To open the Component window, double-click Text Data Sink.
2. In the Column Names pane, right-click the Regenerate the column definition icon.

Working with Fixed-Length Files

For fixed-length file types, you must create the columns and provide the position parameters for each column.

Adding Columns to the Output

Use the component window to add columns to the output.

1. To open the Component window, double-click Text Data Sink.

2. In the Column Names pane, click the Insert a New Attribute icon available. You can edit the name of the generated column.

Removing Columns from the Output

Use the component window to remove columns from the output.

1. To open the Component window, double-click Text Data Sink.
2. Select the column and click the Remove an attribute icon.

Text Data Sink Properties List

The Text Data Sink Properties list identifies the required and optional properties of the Text Data Sink component.

Table 37. Required Properties

Property	Description
Text Destination	Specify the output file. Text Data Sink prompts you for the destination file when you add a component to a project. To specify a destination file, click the **Destination File** icon in the Properties window, and select an existing file, or type the full path and file name to create one during project execution.
Columns	Define columns for data in the source file. If there is a property value defined, the Columns value reflects the port structure or attribute values you defined on the Component window.

Table 38. Optional Properties

Property	Description
Row Delimiter	Specify how each row is delimited: • Position (fixed-line position) • LF (line feed) • CR (carriage return) • CRLF (carriage return followed by a line feed) Alternatively, you can enter a different delimiter.
Row Length	If you have selected Position as the Row Delimiter, specify the number of characters in each fixed row.

Property	Description
Column Delimiter	Specify how columns are delimited: • Position (fixed-column positions) • Tab • Comma • Semicolon Alternatively, you can enter a different delimiter.
Column Quote	Specify how you want the values in the output file to be quoted (delimited files only): • None • Single quote • Double quote Alternatively, enter a different quote character or string

Property	Description
Fixed by Bytes	Specify how to interpret the values provided for line length, column start, and column end: • Not selected (default value) – values are interpreted as number of characters. • Selected – values are interpreted as number of bytes. For example, suppose your source file includes binary: **0x61 62 63 d6 d0 ce c4 61 62 63 64 65** and has these characteristics: • File Type – fixed (variable line) • Encoding – GB2312 • Row delimiter – '\n' • Column definition – column1: 1 – 7; column 2: 9 –10 If you select Fixed by Bytes: • Column 1 displays the first 7 bytes, the binary of which are **0x61 62 63 d6 d0 ce c4**. • Column 2 displays the 9th and 10th bytes, the binary of which are **0x62 63**. If you do not select Fixed by Bytes: • Column 1 displays the first 7 characters, the binary of which are **0x61 62 63 d6d0 cec4 61 62**. • Column 2 displays the next 2 characters, the binary of which are **0x64 65.** **Note:** 0xd6d0, c4c4 represents 2 Chinese characters in GB2312.
Encoding	Set the current character encoding.
Append Column Delimiter	Select if you want the column delimiter to be appended to the end of a row. Selecting this option enhances the performance while loading output files into IQ.
Column Header	Write the column names to the file.
Header	Create a report header to write to the file. Text Data Sink writes the header before it writes the incoming data. Enter the header text; you can use SBN.
Append Data	Append incoming data to the destination file. If you do not set this value, Text Data Sink overwrites any existing data in the destination file.

Property	Description
Write Block Size	Specify the number of records that Sybase IQ InfoPrimer writes to the file in a single write operation.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Running the Text Data Sink Demos

Sybase IQ InfoPrimer includes several demonstrations for the Text Data Sink component. These demos are available as Flash demos and as sample projects in the demo repository.

1. Select **Help** > **Component Demonstrations** > **Destination** > **Text Data Sink**.
2. To access the sample projects, in the Navigator, select **Repository** > **TRANSFORMER.transformer. Repository** >**Projects**. Then select:
   - **Demo XML via SQL Data Provider**
   - **Demo Text Data Sink Delimited/Fixed**

Loader Components

Loader components help to load data from a source database or a file into the IQ database, without performing any transformation.

IQ Loader File via Load Table

The IQ Loader File via Load Table component loads data from a file into a target IQ database using an automatically generated **LOAD TABLE** statement.

This self-contained component operates as a datasource when it reads from source files and a data sink when it writes to a Sybase IQ database. It does not have input and output ports, therefore, there is no need to create one component for performing the read from the source file and another component for invoking **Load Table** in Sybase IQ. Sybase IQ InfoPrimer automatically generates **Load Table** statements that extract data from a delimited text file and loads the data to Sybase IQ.

Configuring IQ Loader File Via Load Table Component

Add and configure a IQ Loader File Via Load Table component to your project.

1. Drag the **IQ Loader File via Load Table** into the Design window.
2. In the Database Configuration window, add the connection parameters for the target IQ database.
3. Select or enter the **Destination table**.

4. Click **Finish.**

See also

- *IQ Loader File via Load Table Properties List* on page 158

Working with the Text Source Property Window

The Text Source property window lets you define the structural properties of data in the source file.

It includes:

- File Content pane
- Properties pane

  Note: When you select a source file, the file path shown in the Text Source field is the path of the machine where Sybase IQ InfoPrimer Development is running. If your grid engine is running on another machine, select a local file, save, and close the window. Reopen the window and replace this path with the file path of the machine on which your grid engine is running.
- Preview pane

Enabling Client-Side Load Support

You can load data into the Sybase IQ table, from files that are located on a different host machine than Sybase IQ. You need not install Sybase IQ InfoPrimer Development and Sybase IQ on the same machine; Sybase IQ InfoPrimer Server and Sybase IQ can communicate in a networked environment, allowing you to bulk-load from a remote machine in a single step.

To support client-side load support:

- Install the Sybase IQ 15.0 client on the same machine as Sybase IQ InfoPrimer Server.
- Install the Sybase SQL Anywhere 11 ODBC driver on the same machine as Sybase IQ InfoPrimer Development and Sybase IQ InfoPrimer Server.
- The target IQ database version must be Sybase IQ 15.x.
- On each Sybase IQ 15.x server, enable the **allow_read_client_file** and **allow_write_client_file** options. To set these options:
  1. From Sybase Central, connect to the Sybase IQ 15.x server.
  2. Right-click the Sybase IQ server database name and select **Options**.
  3. Select **allow_read_client_file** and **allow_write_client_file** options and change their values to on. By default, the value is off.
  4. Enable the **allow_read_client_file** server option property using the **isql** or **dbisql** utility:

     ```
     set option allow_read_client_file=on
     GRANT READCLIENTFILE TO <group | user>
     ```

Once you have completed these prerequisites, select **Use IQ Client Side Load** in the Properties window of the component. Also, select ODBC as the interface, or you may encounter errors loading data from remote host machines. Client-side loading works only with ODBC when the ODBC driver being used is an IQ 15.0 ODBC driver.

Note: If you select **Use IQ Client Side Load** to enable bulk loading of data into the IQ database from files located on client machines, provide a file path name instead of a pipe name in the Load Stage property field. Client-side loading is not supported using Load Stage pipe names.

You are required to make some additional configurations to enable support for multiplex execution by using multiple writers for loading data to IQ.

See also

- *Multiple Writers Configuration for Loading Data* on page 128

IQ Loader File via Load Table Properties List

Required and optional properties of the IQ Loader File via Load Table component.

Table 39. Required Properties

Property	Description
Interface	Identify the method or driver to use to connect to the datasource.
Host Name	Identify the datasource. The options that appear in the Host Name list depend on the interface you select.

Table 40. Optional Properties

Property	Description
User and Password	Identify an authorized database user and protect the database against unauthorized access.
Destination	Select the destination table from a set of existing tables.
Key	Select target key attributes to identify the records on Upsert or Delete operations. If no key is selected, the interface works with the primary key information, which is delivered from the DB host. An error appears if no primary key information is available.

Property	Description
Function	Select one of these load functions: • Insert – to load records directly from the file into the selected target table. • Upsert – to update the existing records and insert the new records. When you select Upsert, the existing records are replaced and not updated on an attribute level. You can use the Key property to specify the target attributes to identify the records you want to update. If you have selected the **Truncate** option, all records are removed from the target table before loading. The Insert and Upsert function loads all records to the target table directly.
Use Binary Load File	Load data from an IQ binary load file. **Note:** If you have selected the Use Binary Load File property, you can only define the source file path in the Text Source property. You cannot specify any other properties.
Text Source	Identify the text file to use as the datasource. From the Properties window, click the Text Source icon, select the file, and specify the format.
Use IQ Client Side Load	Bulk load data from files located on remote host machines into a target IQ database, using the **LOAD TABLE** statement.
Row Delimiter	Specify how each row is delimited: • LF (line feed) • CR (carriage return) • CRLF (carriage return followed by a line feed) Alternatively, you can enter a different delimiter.
Column Delimiter	Specify how columns are delimited: • Tab • Comma • Semicolon • Pipe Alternatively, you can enter a different delimiter.

Property	Description
Load Script	The **LOAD TABLE** statement is generated at runtime based on the component settings, if this property is empty. To use a customized script, right-click the component and select Generate Load Script. The **LOAD TABLE** script is generated for Insert. After you generate the script, you can click Load Script and edit the script. **Note:** If a custom Load Script is provided, the Function property is ignored.
Truncate	Remove all records from the destination table before the load.
Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;). **Note:** If you have selected the Truncate option, all records from the destination table are deleted before the preprocessing SQL statements is executed.
Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Database	Identify the database to use as datasource. If you select this option, you must also select an appropriate interface, and in some cases, specify an appropriate user ID and password.
Schema	Identify the schema or owner you want to use as datasource. The objects that appear are restricted accordingly and new tables are created in the schema you specify.
Database Options	Set options that override performance defaults and control the behavior of some transactions.
Null Indicator	Specify the string that represents null values in the source file.
Skip Rows	Specify the number of rows to skip at the beginning of the input file for a load. The default is 0.
Parallel format	Allow the **LOAD TABLE** command to run in parallel. To use this option, all columns, including the last, must be delimited by a single ASCII character.
Strip	Strip trailing blanks from values before they are inserted. This applies only to variable-length nonbinary data.

Property	Description
Byte Order	Specify the byte ordering during reads. This option applies to all binary input fields. If none are defined, this option is ignored. Sybase IQ InfoPrimer always reads binary data in the format native to the machine it is running on (default is NATIVE). You can also specify: • HIGH when multibyte quantities have the high-order byte first. • LOW when multibyte quantities have the low-order byte first.
Block Size	Specify the default size, in bytes, in which input should be read.
Limit	Specify the maximum number of rows to insert into the table. The default is 0 for no limit.
ON File Error	Specify the action Sybase IQ should take when an input file cannot be opened, either because it does not exist or because you have incorrect permissions to read the file. For all other reasons or errors, it aborts the entire insertion. Specify one of the following: • ROLLBACK (the default) – aborts the entire transaction. • FINISH – finishes the insertions already completed and ends the load operation. • CONTINUE – returns an error but only skips the file to continue the load operation. You cannot use this option with partial-width inserts.
Word Skip	Allow the load to continue when it encounters data longer than the limit specified when the word index was created.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.

Property	Description
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify “00:00:00.000” as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.
Use IQ Multiplex	Support multiplex execution by using multiple writers to load data into the IQ database.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

IQ Loader DB via Insert Location

The IQ Loader DB via Insert Location component loads data from the source database into the target IQ database using the **Insert Location** statement.

This self-contained component operates as a datasource when it reads from a source database and a data sink when it writes to a Sybase IQ database. It does not have input ports and output ports, therefore, you do not have to create one component for performing the read from the source database and another component for invoking **Insert Location** in Sybase IQ. Sybase IQ InfoPrimer automatically generates **Insert Location** statements to transfer data from the source database to Sybase IQ. **Insert Location**:

- Allows you to move columns from Sybase IQ versions earlier than 12.0 to version 12.0 and later.
- Provides optimized load to Sybase IQ from Adaptive Server Enterprise or Sybase IQ.
- Also works with Sybase Enterprise Connect™ Data Access (ECDA) to load Sybase IQ from Oracle, IBM DB2, and Microsoft SQL Server. Sybase IQ InfoPrimer supports Sybase ECDA 15.0 with IBM DB2 9.1, Oracle 10g, and Microsoft SQL Server 2005.

  For Sybase ECDA documentation, see the Sybase Product Documentation Web site at *http://www.sybase.com/support/manuals*.

Note: Sybase is the only interface supported by the IQ Loader DB via Insert Location component.

Configuring the IQ Loader DB via Insert Location Component

Add and configure a IQ Loader DB via Insert Location component to your project.

1. Drag the **IQ Loader DB via Insert Location** component into the Design window.
2. Enter IQ database connection properties for the destination database.
   - **Host**
   - **User**
   - **Password**
   - **Database** – select the database to use as the destination database.
   - **Schema** – select schema/owner to restrict the objects that appear and create new tables in that schema.

   Click **Processing** to enter preprocessing and postprocessing SQL statement on the IQ destination database.

   Click **Logon** to view the list of available tables.

   Click **Next.**
3. Enter connection information for the source database and select the tables to transfer.
   - Select **Use remote server definition for accessing source database** to get data and metadata from the source. Select this option only if you have used the **Create Server** command to define the source server as a remote server on the destination IQ database. If you do not select this option, you are directly connected to the source data using the configuration information provided in the `.INI` or `interfaces` file.
   - **Host**
   - **Database**
   - **Schema** – select schema/owner to restrict the objects displayed and create new tables in that schema.
   - Click **Processing** to enter preprocessing and postprocessing SQL on the source database.
   - Select **Create Target Tables** to create the destination tables if they do not exist.
   - Select **Continue on Error** if you want processing to continue even if an error occurs when loading data into a database.
   - Select **Encrypted Password** to transmit the password in encrypted format.

   Note: When used as a remote server, Sybase IQ does not support password encryption.

   - Select **Use IQ Multiplex** to support multiplex execution by using multiple writers for loading data to IQ. Select this option if more than one table is being migrated to the IQ database.
   - Select **Lock Table** to lock the target table in Exclusive mode and prevent it from being updated by concurrent transactions. If selected, no other transaction can execute queries or perform any updates against the locked table. The Lock Table option also queues multiple projects that load the same table in Sybase IQ.

  If you select this option, you must also specify the maximum blocking time that the project should wait before acquiring the lock.
- Enter a network packet size in the `Packet Size` field.
- Enter a value in the `Limit Rows` field.
- Specify the number of rows to skip at the beginning of the input tables for a load in the `Skip Rows` field.
- Click **Logon** to view the list of available tables for the specified database. By default, each table is selected for transfer. Unselect the Transfer option for tables you do not want to transfer. You can also choose one or multiple table rows, right-click and select **Exclude**. To include a table for transfer, right-click and select **Transfer.**

  Alternatively:
  - Click **Exclude all objects from transfer** to exclude all tables.
  - Click **Include all objects in transfer** to include all tables.
- Click **Next.**

4. Verify the source tables. They should be in the following format:
   ```
   source_schema.source_table
   ```
5. Select the destination tables. There should be a one-to-one mapping (one source for each destination) between sources and destinations.
6. Click "Truncate Destination" to delete all existing data rows from the destination table.
7. Click Next.
8. Review the load configuration summary. Click Finish.

See also

- *Multiple Writers Configuration for Loading Data* on page 128

Setting Locales for Insert Location Statements

When you execute **Insert Location** statements, Sybase IQ loads the localization information needed to determine language, collation sequence, character set, and date/time format. If your database uses a non-default locale for your platform, you must set an environment variable on your local client to ensure that Sybase IQ loads the correct information.

If you set the LC_ALL environment variable, Sybase IQ uses its value as the locale name. If you do not set LC_ALL, Sybase IQ uses the value of the LANG environment variable. If neither variable is set, Sybase IQ uses the default entry in the locales file. See *Sybase IQ 12.7 System Administration Guide > International Languages and Character Sets > Setting locales*.

IQ Loader DB via Insert Location Properties List

IQ Loader DB via Insert Location properties list identifies the connection parameters and other items you must define on the IQ Loader DB via Insert Location component window.

Table 41. Required Properties

Property	Description
IQ Host Name	Specify the host where the Sybase IQ target is running.
IQ User	Specify an authorized IQ user to protect the database against unauthorized access.
IQ Password	Specify a password to protect the database against unauthorized access.
Source Host Name	Specify the datasource.
Source Database	Specify the source database.
Source Transfer List	Specify the schema qualified source table name and the target table name. In the target truncate column, specify 1 to truncate the target table, otherwise enter 0.

Table 42. Optional Properties

Property	Description
IQ Database	Specify the IQ destination database.
IQ Schema	Specify the IQ schema/owner to restrict the objects shown and create new tables in that schema.
IQ Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
IQ Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Use Remote Definition	Select this option only if you have defined the source server as your remote server on the destination IQ database using the **Create Server** command. If you do not select this option, you are directly connected to the source data using the configuration information provided in the `.INI` or `interfaces` file.
Source Schema	Specify the schema/owner to restrict the objects displayed.

Property	Description
Source Pre Processing SQL	Create a script that runs during component initialization. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Source Post Processing SQL	Create a script that runs after all components execute. Scripts can include one or more SQL statements. If you use multiple statements, separate them with a semicolon (;).
Function	Select one of these load functions: • Insert – to load records directly from the source into the selected target table. • Upsert – to update the existing records and insert the new records. When you select Upsert, the existing records are replaced and not updated on an attribute level. The table you are working with must have a predefined primary key. If you have selected the Truncate option, all records are removed from the target table before loading. The Insert and Upsert function loads all records to the target table directly.
Create Target Tables	Create the destination tables if they do not exist.
Continue on Error	Continue execution even if an error occurs when loading data into the database.
Limit Rows	Specify the maximum number of rows to insert into the table. The default is 0 for no limit.
Skip Rows	Specify the number of rows to skip at the beginning of the input tables for a load. The default is 0.
Encrypted Password	Transmit the password in encrypted format.
Packet Size	Specify the network packet size.
Load Script	The **Insert Location** statements are generated at runtime based on the component settings, if this property is empty. To use a customized script, right-click the component and select **Generate Load Script**. The **Insert Location** script is generated for Insert. After you generate the script, you can click **Load Script** and edit the script. **Note:** If you have provided a custom Load Script, the Function property is ignored.

Property	Description
Use IQ Multiplex	Support multiplex execution by using multiple writers to load data into the IQ database.
IQ Lock Table in Exclusive Mode	Lock the target table to prevent it from being updated by concurrent transactions. When an exclusive table lock is applied, no other transaction can execute queries or perform any updates against the locked table. The IQ Lock Table in Exclusive Mode option also queues multiple projects that load the same table in Sybase IQ.
Wait Time for IQ Lock Table	Specify the maximum blocking time that the project should wait before acquiring an exclusive lock. Specify the time argument in the format **hh:mm:ss.sss.**. When you do not enter a time argument, the server does not wait. When you specify "00:00:00.000" as the time argument, the server waits indefinitely until an exclusive lock is available or an interruption occurs.
Transactional	All work performed by the component, including preprocessing SQL statements and postprocessing SQL statements, is done in a single database transaction that is committed when the project finishes normally. Select this option to roll back the transaction, if this component encounters an error.

Accessing the ETL Demonstration Projects and Jobs

Connect to the demonstration repository to view examples of ETL projects and jobs.

1. In Windows, select **Start > Programs > Sybase > Sybase IQ InfoPrimer Development 15.3 > Sybase IQ InfoPrimer Development**.

   The Login window shows the configuration for the demonstration repository, which includes sample projects. Each sample project shows how to use a component or implement a scenario.

   - Connection – `Repository`
   - Client – `transformer`
   - Client user name – `TRANSFORMER`
   - Password – `transformer`

     These values are automatically set the first time you log in. On subsequent logins, you might need to select or enter this information.

   Click **Logon.**

Note: Sybase does not recommend you to use the demonstration repository in production environments.

2. In the Navigator, click **Repository** > **TRANSFORMER.transformer.Repository** > **ETL Projects** folder or the **Jobs** folder to open the list of available ETL projects and jobs.
3. Double-click an existing project name to open it, or to create a new ETL project, right-click **ETL Projects** and select **New**.

Restoring ODBC Datasources

Restore the initial set of datasources that are created by the Sybase IQ InfoPrimer installer if you are logging in to the demonstration repository. If repository datasources are lost for any reason, Sybase IQ InfoPrimer cannot access or execute the demonstration projects until you restore them.

Note: This procedure restores only the datasources; it does not restore the demonstration repository and databases to their initial state.

1. Configure the ODBC user datasource, when the Sybase IQ InfoPrimer is installed for a single user:
   a) Select **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

      On Windows 7 and 2008, select **Start** > **Control Panel** > **System and Security** > **Administrative Tools** > **Data Sources(ODBC)**.

      Note: To administer datasources on a 64-bit machine, run `odbcad32.exe` from `C:\WINDOWS\SysWOW64`.
   b) Click **Add**.
   c) Select **SQL Anywhere 11**. Click **Finish**.
   d) Enter `DEMO_Repository` as the ODBC datasource name.
   e) Click the **Login** tab and enter `dba` as the user ID and `sql` as the password.
   f) Click the **Database** tab and in the Start line field, enter the installation location. The default is `C:\Sybase\IQIPD-15_3\dbeng11.exe`.

      Note: IQ InfoPrimer Development runs `dbeng11.exe` for configuring the IQ InfoPrimer repository. `dbeng11.exe` is a single-user database that cannot be accessed from other machines. To access the repository from another machine, specify `dbsrv11.exe` instead of `dbeng11.exe` in the Start line field.
   g) Edit the Database file field:
      - If the application is installed for a single user in the default location, enter `C:\Sybase\IQIPD-15_3\demodata\demo_rep.db`
      - If the application is installed for all users:
        - For Windows XP or 2003 – enter `C:\Documents and Settings\<login user>\Application Data\SYBASE\Sybase IQ`

`InfoPrimer Development\<product version>\Demodata\demo_rep.db`.
- For Windows 7 and 2008 – enter `C:\Users\<login user>\AppData\Roaming\SYBASE\Sybase IQ InfoPrimer Development\<product version>\Demodata\demo_rep.db`.

h) Click the **ODBC** tab and click **Test Connection**.

2. Set up the repository connection:

a) Select **File** > **Open Repository**.

b) Select **Repository** from the Connection list and choose:
- **Edit** or,
- **Add**, and enter a name for the connection.

c) Select **ODBC** from the Interface list.

d) Select **DEMO_Repository** from the Host list.

e) Click **Save**.

3. Configure the additional ODBC user datasources required by the projects in the demo repository:
- Driver – SQL Anywhere 11
- Name – ETLDEMO_DWH; database – demo_dwh.db
- Name – ETLDEMO_GER; database – demo_ger.db
- Name – ETLDEMO_US; database – demo_us.db

The database files for these user datasources are also located in the `Demodata` folder of the installation directory, when installed for a single user, or in the user data directory, when installed for all users.

Simulating ETL Projects Interactively

Start and simulate an ETL project interactively.

To run the project interactively, select **Run** > **Trace**. You can stop the simulation at any point, and select **Run** > **Step** > **Step Through** to manually step through the remaining project components.

1. To start a simulation, click **Start** on the toolbar:
- All components of the project are initialized.
- All connections within the project are validated.
- All preprocessing SQL statements in the projects are executed.
- Data for all static lookup components is retrieved and cached. Any change of data in lookup tables during simulation is not reflected in the simulation process.

2. Select a component and click the **Step** icon on the toolbar to execute the component.

"Stepping a component" means executing or processing a single component. The data records that are processed during a single step are the records currently populating the IN-ports of the component.

If a component is stepped multiple times and no other components are stepped in between, the number of records received or forwarded remains constant. Many components can be stepped from both inside the component and outside in the project view.

3. View the data flow on the connecting link or within the component.

   - To view data throughout transformation, examine the link between components or the ports of a component. Other components such as the Data Calculator and the Data Splitter include built-in preview capabilities.
   - To view data on the connecting link, right-click and select **Preview**.
   - To view data currently at the port, right-click the port, and select **Preview.**

   Note: The Preview option is disabled when there are no processed records or when no simulation data is available.

   - To view data from inside the component, double-click the component, or click the Rule icon in the Property window. Use this option to see the impact of transformation rules from within components, such as the Data Calculator or the Data Splitter.

4. Modify and initialize the component.

   After you modify a component, you can reinitialize the component to continue simulating the data flow. You need not restart a complete simulation for the current project.

   a) Double-click the component to modify its properties in the Properties window.
   b) Save the changes.
   c) Right-click the component and select **Initialize**.

5. When all data has been processed, select one of:

   - **Execute Post-Processing as for successful execution** – to commit all tasks performed by the transactional components and to reset the project to its initial state.
   - **Execute Post-Processing as for failed execution** – to rollback all tasks performed by the transactional components and to reset the project to its initial state.

   Click **Yes** to confirm resetting of the interactive trace. This clears all port buffers, releases temporary tables, and closes all database connections and temporary files.

   If you click **No**, all open database connections and port buffers are retained. You can inspect, reconfigure, and then restep individual components.

   To reset trace to commit or roll back data for the transactional components, click **Reset** on the toolbar, or select **Run > Reset**. Click **Yes** to confirm.

Setting the Trace Delay

You can control the simulation rate by setting the simulation interactive trace delay option.

1. Select **File > Preferences > Engine**.
2. Modify the value in the Rate of simulation field. You can enter values between 10 and 9999 milliseconds. The default value is 250 milliseconds.

Viewing Current Mappings

The Mapping Definition window shows the current mapping between attributes of adjacent input structures and output structures.

1. Right-click the connecting link, and select **Mapping.**
2. In the Mapping Definition window, select:
   - **Display structure** to view all attributes of the connected port and their current mappings.
   - **Display structure and values** to view the fields as well as the values of the current record. This view shows the current content of the port connecting to the link. If the port contains no data, only the port structure is shown in this window. You can populate data in a port by stepping through your project until you reach the port.

Applying Automatic Mappings

Use the **Mapping** menu to create and apply automatic mappings.
Select one of the predefined mapping sequences.

- **Create mapping by order** – sequentially maps the port attributes of the IN- structures and OUT- structures. If the number of attributes is different on both sides, some of the port attributes are not mapped.
- **Create mapping by name** – maps port attributes of the IN- structures and OUT- structures, according to their names.
- **Create mapping by name case-sensitive** – maps the port attributes of the IN- structures and OUT- structures, according to their case-sensitive names.
- **Create mapping by prefix** – maps the port attributes of the IN- structures and OUT- structures by name, ignoring the specified prefixes.
- **Create mapping by best match** – maps the port attributes of the IN- structures and OUT- structures that sound alike.

Applying Manual Mappings

Manually create a single mapping using the Mapping menu.

- Select a connection point and drag it to the connection point of a port attribute to create a mapping.

- To change a current mapping, select the mapping line at the connection point and drag it to an unmapped port attribute.
- To delete a single mapping, right-click the mapping, and select **Delete**.
- To delete all mappings of a link, select **Remove All** from the Mapping menu.

Viewing Mapped Attributes

By default, the Mapping Definition window displays all the port attributes of the IN structures and OUT- structures.

To view only the mapped attributes, click **Display only mapped attributes** on the toolbar.

Enabling Synchronized Attribute Scrolling

You can enable synchronized scrolling between the attributes of the IN- structures and OUT-structures.

Click the **Synchronize attribute scrolling** icon on the toolbar.

Managing Port Attributes

You can add and delete port attributes, or modify the settings or existing attributes in the Structure Viewer window.

Adding Attributes to the Port Structure

Add attributes to the port structure using the Struture Viewer window.

1. In the Design window, right-click the port, and select **Edit Structure**.
2. In the Structure Viewer window, select **Actions > Add**, or right-click the attribute and select **Add**.
3. Enter a name for the attribute. The names for port attributes must start with an alphabet character and can contain only alphanumeric characters. You cannot have reserved JavaScript keywords as names.

   Select **Populate Attribute** to add the attribute to multiple port structures. The new attribute is added to all port structures participating in the selected connections and is automatically mapped. Click **OK.**
4. Specify the other details.

See also

- *Variables* on page 176

Deleting Attributes from the Port Structure

Remove attributes from a port structure using the Struture Viewer window.

1. In the Design window, right-click the port, and select **Edit Structure**.

2. In the Structure Viewer window, select **Actions > Remove**, or right-click the attribute and select Remove.

Modifying Port Attributes

Use the Struture Viewer window to edit attributes for a port structure.

1. In the Design window, right-click the port, and select **Edit Structure**.
2. In the Structure Viewer window, change the attribute settings, and click **Save**.

Modifying Datatypes

Modify datatypes of a record structure, to modify the internal logical representation that Sybase IQ InfoPrimer uses for the record structure during transformation.

This does not change the data structure definition of the source or destination tables. Make sure that the data structure of the final Data Sink is compatible with the content you are generating.

Viewing a Simulation Flow

View a simulation flow.

After you start simulation:

- A green dotted box indicates the active component and moves with each step from one component to the next.
- The number of records appears on the link, which follows the box movement.

The number of records being processed within each single step depends on the current value of Read Block Size of the previous component with a Read Block Size property.

It is useful while performing a simulation to select a small number of records. A large number for Read Block Size can significantly enhance performance when a project executes.

Stepping from Current and Selected Components

When a simulation starts, the first component to be executed is indicated with a dotted green box. When you step through the project without making modifications, the box moves from component to component, displaying the success or failure icon until the end of the simulation.

You can select a component other than the current one to inspect or change its properties. The selected component is indicated by a solid green box.

The current component is executed next, when you click **Step** on the toolbar or select **Run > Step**. To inspect or change a component that is different from the current component, click it. The solid green box highlights the selected component.

After making changes, resume the simulation from either the selected or the current component:

- To resume simulation from the selected component, right-click and select **Step.**
- To resume simulation from the current component, click **Step** on the toolbar.

Note: If you right-click an unprocessed component, and select **Initialize and Step**, the component is initialized and stepped, and the next component to be processed is highlighted.

Forwarding and Backward-forwarding Components

The visual flow of the simulation as indicated by the box is straightforward in most projects; the box moves from one component to the next. However, the flow of a project simulation does not necessarily progress in only one direction; flow direction depends on the components used within the project.

Forwarding components, such as Data Calculator and Character Mapper, receive a number of records, apply transformation to those records, and forward the records. The number of records processed in a single step is determined exclusively by the number of records received from the preceding component.

Other components override previous Read Block Size settings. The Staging component works on the entire result set of the data stream, as defined with the query of the Data Source component. The component does not process and forward any data records until the entire result set is delivered to the IN-port. The Staging component uses its own Read Block Size property to resize the number of records forwarded with the next step.

Previewing Data from Multiple Locations

Use the Content Browser to view the data currently available at the selected location.

Right-click any connecting link, port, or component, and select **Preview** to open the Content Browser window.

Note: The Preview option is not available when there are no processed records or when no simulation data is available.

The Content Browser window displays the data currently available at the selected location. It includes tabs that allow you to simultaneously display multiple previews from multiple locations. You may find it useful to preview the content of both the input port and output port of a component.

To save the shown data to a definition file, click the **Export data** icon on the toolbar. Specify the options for exporting data.

Partially Executing or Initializing During Simulation

Restarting an entire simulation after making modifications to a single component can be time consuming, especially if your project has a large number of input records in a project that consists of large number of components.

To make multiple steps through a project to your point of interest, select the component and choose **Run** > **Step through** or **Run** > **Start**.

Simulating Up to a Certain Component

Validate your current project by starting from a component somewhere in the middle of a project.

Select the component, and then select **Run > Start Through**.

The simulation starts the current project, processes all components between the current and the selected component, and processes the selected component.

Impact of Read/Write Block Size

The number you enter as the Read Block Size defines the number of records fetched by the component during a single simulation step. Set the Write Block Size to define the number of records to be written.

Most data provider components possess a Read Block Size property. You can customize the Write Block Size for most of the data sink components. You can customize both reading and writing values for transformation components, such as the Staging component.

Note: The Block Size property is evaluated during project simulation as well as project and job execution. A small number might be suitable for simulation purposes, but slows execution. In simulation, the Block Size is restricted to 32K.

Controlling Multiple Data Streams

Set up a single project that has multiple unconnected data streams. You cannot predict the order in which the streams are processed. In general, most projects consist of a single stream of components connected through links.

If you use multiple data streams, Sybase recommends that you design a project for each data stream so that all components within a project are connected to each other. This lets you control data streams by connecting projects to form a job process flow.

Customizing SQL and Transformation Rules

When setting up an ETL project or job, you can customize SQL queries for setting up source components, SQL commands for processing tasks, Expressions, conditions, and procedures to manipulate the transformation process.

The format of SQL commands depends on the database system that is connected to the component. However, the format of the transformation language supported by Sybase IQ InfoPrimer (JavaScript) does not change, regardless of the source or target system you use in your projects.

You can include JavaScript expressions in square bracket notation (SBN), which considerably reduces your customization efforts. SBN expressions can be part of component properties (if the **Evaluate** option is activated for the specific property), like SQL statements, or any processing commands. An SBN expression is evaluated at runtime, as opposed to constant values, which are defined at design time.

See also

- *Evaluating SBN Expressions* on page 67

Expressions and Procedures

An expression is a combination of identifiers and operators that can calculate a single value. A simple expression can be a variable, a constant, or a scalar function. You can use operators to join two or more simple expressions into a complex expression.

Examples of expressions are:

```
'Miller'
uConcat("Time ", "goes by")
(uMid(SA_ORDERDATE, 1, 10) >= '1998-01-01')
```

A procedure is a programming unit that includes expressions, statements, and control structures. You can write procedures in JavaScript, for example:

```
if (IN.PR_PRICE < 250)
       OUT.PR_GROUP2 = 'low end' ;
else {
    if (IN.PR_PRICE < 1000)
       OUT.PR_GROUP2 =  'mid range';
    else
       OUT.PR_GROUP2 = 'high end';
}
```

Variables

A variable is a symbolic name for a value.

There are two basic properties of a variable:

- Scope, which specifies the range of the environment in which the variables can be referenced.
- Datatype

There are two kinds of variables:

- Port variables
- Component variables

Port Variables

The values of the port structure are referenced as port variables within a component. There are automatic port variables for both input ports and output ports. Port variables are valid within the component, and they inherit the name and datatype of the port structure. The name of the variable is prefixed with "IN." for the input ports and "OUT." for the output ports. input port variables are read-only, but output port variables can be written.

This example uses port variables in an expression:

```
uUpper(IN.CU_NAME)
```

This example uses port variables in a procedure:

```
OUT.CU_NAME = uUpper(IN.CU_NAME);
```

Component Variables

Component variables are associated with component properties and represent the current evaluated property content. You can reference these variables inside the component. The name of the variable is prefixed with "REF." For example:

```
uIsNull(REF.Host)
```

To provide flexibility in transformations, all port and component variables internally use the datatype `string`. If you use numeric values, this may result in unexpected behavior.

If multiplied by 1, the numeric value of a string variable is used in a calculation:

```
IN.Margin="2", IN.Price="10"
IN.Margin>IN.Price  - returns TRUE
```

To enforce a numeric comparison, use:

```
IN.Margin*1>IN.Price*1  - returns
FALSE
```

Do not use reserved JavaScript keywords for port and component variable names:

Reserved JavaScript Keywords				
break	case	catch	continue	default
delete	do	else	finally	for
function	if	in	instanceof	new
return	switch	this	throw	try
typeof	var	void	while	with
abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile	const	export		

Functions

Sybase IQ InfoPrimer includes a complete set of functions and operators that supports Unicode character sets.

You can recognize Sybase IQ InfoPrimer functions by the prefix **u**, for example, **uConcat()**.

Square Bracket Notation

Most component properties can contain SBN expressions that are evaluated before being used by the IQ InfoPrimer Server on project configuration, simulation, or execution. An SBN expression is a JavaScript expression surrounded by square brackets [..]. You can use multiple SBN expressions within a single string value.

You can use SBN expressions in component properties like:

- SQL queries
- Processing SQL statements
- Transformation rules
- File names
- Path definitions
- URLs

Examples

A literal is a string surrounded by quotes. If you use SBN in a literal, the SBN is evaluated first.

```
‘Arrival Date: [uDate(‘now’, ‘localtime’)]’
```

This expression specifies the path of a file in the Text Data Provider:

```
 [uSystemFolder(‘APP DEMODATA’)]\PRODUCTS.TXT
```

Note: In the Property window of components, the `Eval` column indicates whether a value entered is evaluated to resolve SBN expressions. For many property items, the Eval column is optional. To toggle the Eval check box, right-click the property item and select **Evaluate.**

Working with SQL Properties

Nearly all components with database connectivity support custom SQL statements executed in different phases of the transformation. There are two basic types of SQL properties, queries and scripts.

For both types:

- Any SQL accepted by the connected database system is allowed. Using SQL92 allows you switching to different database systems without changing the statements.
- SBN expressions are allowed.

See also

- *Square Bracket Notation* on page 178

Queries

SQL queries are used for all components that extract data, mainly the Data Provider and Staging components. The columns of the query result set define the respective output port structure.

The SQL query result set must contain at least one column. This is an example of a SQL query:

```
SELECT cu_no, cu_name FROM customers
```

Scripts

A SQL script consists of one or more SQL statements that do not return any data. For example, there are properties allowing you to execute SQL statements during initialization (preprocessing) of a component or after completion (postprocessing) of the project.

Some considerations:

- SQL statements must not return output after execution.
- You can enter multiple SQL statements in a SQL property by using a semicolon as a statement delimiter.
- When using stored procedures in a SQL property, include the **call** keyword before the stored procedure name. For example, **call** my_proc(); where my_proc() is the name of the stored procedure.

This is an example of a SQL script:

```
DELETE FROM products;
UPDATE customers
SET cu_desc = 'valid';
```

Using the SQL Property Window

To modify the value for a SQL property of a component, click the respective **Edit** icon in the Properties window.

In the SQL Property window, you can:

- Manually enter statements
- Use the Query Designer.
- Look up the database schema.
- Run a query or script.
- Save the SQL property value.

See also

- *Using the Query Designer to Create Queries* on page 188

Entering SQL Statements

You can enter the SQL statements in the **Query** text field of the SQL Property window.

Creating a Query

You can open the Query Designer to graphically construct a query.

1. In the Query Designer, select **View > Generated Query** or click the **Generated Query** tab.
2. Copy the entire or parts of the generated query, and paste it in the **Query** text field of the SQL Property window.

   Note: For query properties, you can click Save to replace the entire contents of the Query text field with the generated query.

See also

- *Using the Query Designer to Create Queries* on page 188

Looking Up the Database Schema

You can select tables and attributes for your statements from the associated database schema by clicking the **Database Lookup** icon. In the Database Lookup window, select the tables and attributes you want to use in the statement and click **OK**.

Executing SQL Statements and Viewing Query Result Sets

To execute the current SQL statements, click the **Execute** icon. For Query properties, you are prompted to specify the number of records to view. The result is shown in the Data Viewer.

SBN Expressions

You can use SBN expressions in SQL properties. Although the specified example is a query, the technique applies to both types.

Assume you want a **SELECT** statement to retrieve a specific customer record.

- You might include a constant customer number for that record:
  ```
  select * FROM CUSTOMERS WHERE CU_NO = '12345678'
  ```
- With SBN, you can use a more flexible approach by assigning the constant value of CU_NO to a custom component property. Assuming that value 12345678 was assigned to a property CustNo, the **select** statement with the dynamic expression looks like:
  ```
  select * FROM CUSTOMERS WHERE CU_NO = '[REF.CustNo]'
  ```
- You can use any of the Sybase IQ InfoPrimer functions inside the SBN expression. The following statement returns the same record using a value of 1234 for CustNo1 and a value 5678 for CustNo2:
  ```
  select * FROM CUSTOMERS WHERE CU_NO = '[uConcat (REF.CustNo1,
  REF.CustNo2)]'
  ```

See also

- *Adding and Removing Custom Properties* on page 68

JavaScript Editor and Debugger

JavaScript is an object-oriented scripting language designed that is embedded into other products and applications. JavaScript allows manipulation of objects to provide programmatic control over them.

JavaScript functionality is enriched by grid functions, which enhance the flexibility of the language. The JavaScript Editor and Debugger let you interactively edit, debug, and execute JavaScript code.

The JavaScript Editor and Debugger is mainly used to set up transformation rules on incoming data.

The JavaScript Editor and Debugger offers:

- Color-coded syntax for better readability
- A watch list to control the assigned values of variables and attributes while you are running or stepping through code
- Multiple user-defined breakpoints to stop code execution at any line positions
- User-defined Go points to arbitrarily choose the position from which code executes
- Step mode to execute code line by line
- Step-over during debugging
- Evaluation of JavaScript expressions
- Verification of the result of code execution

Starting the JavaScript Editor and Debugger

Use the the JavaScript Editor and Debugge to execute and test the scripts using a single input record.

1. Double-click the **Data Calculator JavaScript** component, or click the **Rule** icon in the Property window.
2. Select a row in the Transformation Rule column and click the **Edit** icon.

The JavaScript Editor and Debugger window includes:

- Navigator – consists of the Variables tab and the JavaScript tab. The Variable tab consists of input port and output port variables, temporary, and predefined variables. The JavaScript tab includes all functions, commands, and system variables that can be applied within the procedure.
- Edit/Debug pane – lets you edit the actual code.
- These tabs appear at the bottom of the window:
  - Tasks – displays the results of the validation after your procedure has been compiled.

- Watch List – displays selected variables and their values when stepping through the code during debugging.
- Input Records – displays the content of the current input record. To synchronize input records and output records, click **Start debugging** on the toolbar.
- Output Record – displays the content of the current output record.
- Expression – displays the result of the expression after you enter a JavaScript expression and click **Evaluate** on the toolbar.

  Note: The **Evaluate** button is disabled when you are not in an active debugging session.

Edit and Debug Mode

The JavaScript Editor and Debugger opens in edit mode.

1. To switch to the debug mode, select **Debug** > **Start**.
2. A dark grey background of the edit area indicates the debug mode. To switch to edit mode, click the **Stop debugging and start editing** icon on the toolbar.

JavaScript Editing and Debugging

The JavaScript Editor and Debugger lets you trace the execution of a script. You can step through a code line-by-line or step through from one breakpoint to another. You can check the current value of a variable at any time.

Note: A comment line starts with two forward slashes (//) at the beginning of the line.

To validate JavaScript code, select **Debug** > **Start**. The result of the validation appears in the **Tasks** tab.

Stepping Through the Code

The JavaScript Editor and Debugger works without input data at the IN-port of the component. However, for best results, populate the IN-port with data before using the debugging features.

1. Validate the script or switch to debug mode.

   A green arrow, pointing initially to line 1, indicates the progress of the execution while stepping.
2. Make sure the result message in the Task tab displays `Successful compilation`.
3. To move to the next line, click the **Step** icon on the toolbar.

   At any point during stepping, you can inspect the variable name and the current value by selecting the variable in the Navigator,and right-clicking it.

Adding and Removing Breakpoints

Include breakpoints at selected lines instead of stepping through the procedure line-by-line.

1. Click the line where you want to set or remove a breakpoint.
2. Right-click and select **Add/Remove** breakpoint.

Stepping to a Breakpoint

Step to a breakpoint.

1. Click the **Go** icon on the toolbar for each step.
2. Click the **Go** icon on the last breakpoint to execute the rest of the script.

Monitoring Values in the Watch List

You can use the watch list to monitor the changes of variable values during the code execution. When stepping through the code, you can see any change that occurs to one or more variables in the watch list.

Adding a Variable to the Watch List

Add a new variable to the watch list.

1. Right-click the variable.
2. Select **Add to Watchlist**.

Removing a Variable Form the Watch List

Remove an existing variable from the watch list.

1. In the Watch List tab, select the variable row and right-click.
2. Select **Remove Watch Variable**.

Special JavaScript Features

Use the **Cancel a running script** icon on the editor toolbar to interrupt JavaScript execution.

Creating User-Defined Errors

To enforce an error and interrupt the execution of the project, use the **throw("xx")** function. For example, to stop execution if the name of a product (PR_NAME) exceeds the length of 20 characters, use:

```
if (uLength(IN.PR_NAME) > 20) (
      throw(“Product name exceeds maximum length”);
)
```

Creating User-Defined Functions

You can define functions inside a script and create nested functions. For example, this script results in a value 6 for variable *b*:

```
var a = 2;
var b = 20;
b = IncA(a);
// end
function IncA (a)
```

```
{
     var b = 3;
     a = IncB(b) + a++;
     return a;
     function IncB(b)
     {
          b = b + 1;
          return b;
     }
}
```

Converting Datatypes

All variables in Sybase IQ InfoPrimer are represented as strings. If you work with numeric values, this may result in unexpected behavior. You can use the functions **parseInt()** and **parseFloat()** to convert a string to an integer or a float, for example:

```
var a = "123";
var b = "22";

a > b

will return FALSE while

parseInt(a) > parseInt(b)
returns TRUE.
```

Including Files

To include external files into a script, use the **uScriptLoad("filename")** function. The external file can contain any valid JavaScript constructs, including functions, thus allowing a kind of reusable code, for example:

```
uScriptLoad("C:\scripts\myfunc.js");
var a = 11;
var b = 2;
var c = 0;
b = gcd(a, b);
// gcd function defined in C:\scripts\myfunc.js
```

Creating and Simulating a Sample ETL Project

Create and simulate a sample project that includes one or more data provider, data sink and a data calculator components.

- Data providers provide the data feeding the project data stream
- Data transformers transform or remap field values
- Data sinks write transformed values to their target

Note: You can view results of this section in the Demo Getting Started project in your default repository.

Adding and Configuring a Data Provider

Add the DB Data Provider Full Load component to your project.

- Drag the component from the Source tab of the Component Store to the Design window.
- In the Component Store, right-click the component that you want to add, and select **Add.**
- In the Component Store, double-click the component you want to add.

When you add a component to the Design window, the default configuration of the component is shown.

Note: Properties shown in **bold** in any configuration window are required.

1. Select **ODBC** as the interface.
2. Select **ETLDEMO_US** as the host.

   When you confirm the initial component settings, the settings appear in the Properties window.
3. To define the information to retrieve from the datasource, click the **Query** icon in the Query field.
4. Click the **Query Designer** icon to generate the query.

   Note: You can also manually enter the SQL query.

   The left pane of the Query Designer window lets you navigate the table catalog of the connected database.
5. To add one or more tables, drag the table name onto the Design window, or right-click the table name and select **Add Object to Query**.
6. Click and drag the **`PRODUCTS`** table to the Design window.
7. Click **Save** to close Query Designer and return to the Query window. The **select** query is automatically generated.
8. Click **Execute the Query** icon to run or test the query.
9. Click **Save** to close the Query window.

   Note: When you have successfully configured a component, the color of the ports associated with it change from red or yellow to green.

Adding and Configuring a Data Sink

Add DB Data Sink Insert to your project.

- Drag the component from the Destination tab of the Component Store to the Design window.
- In the Component Store, right-click the component that you want to add, and select Add.
- In the Component Store, double-click the component you want to add.

As soon as you add a component to the Design window, the component displays its default configuration.

Note: Properties shown in **bold** in any configuration window are required.

1. Select **ODBC** as the interface.
2. Select **ETLDEMO_DWH** as the host.
3. Enter `PRODUCTS` in the Destination Table field. Alternatively, click the **Destination table** icon and select **PRODUCTS**.
4. Click **Finish** to confirm your settings.

Your project now consists of two components. The link between the components is created automatically if you selected **Create automatic links when components are added** in the **File > Preference** window. If the line is not automatically created, click the outgoing port and drag it onto the input port of the data sink to create it.

The output port of the DB Data Provider Full Load component and the input port of the DB Data Sink Insert component both display in green. This indicates that both components are configured.

In the Property window for the DB Data Sink Insert component, you can review and set all properties of the selected component.

Reviewing and Defining Attribute Mappings

Learn how to review and define attribute mappings.

1. Right-click the link between the components. The color of the link changes to green.
2. Select **Mapping.**

The mapping between the datasource and the target source is created automatically. To change mappings, select the connecting line and attach it to another connection point.

Note: You can map only to an unassigned target connection point. If all target connection points are already assigned, unassign a connection point by deleting the mapping line that is currently linking to it. Select the mapping line and press the **Delete** key, or right-click and select **Delete**.

Adding a Data Calculator

Add a Data Calculator to your sample project.

1. In the Component Store, click the **Transform** tab.
2. Select and drop the Data Calculator JavaScript component onto the link connecting the existing components. The color of the link changes to blue.

   After releasing the Data Calculator component:

- It is linked with the components to the right and left.
- The Data Calculator window appears.

The Data Calculator window has a Tabular and Graph view:

- Use the Tabular view to enter transformation rules.
- Use the Graph view to visually define the mapping sequence between the input ports and output ports.

3. Click the **Graph** tab. The IN and OUT boxes represent the current structure of the port attributes.
4. Click **Yes** to assign a default mapping by order.
5. Click the **Tabular** tab to return to the tabular view.
6. Change all incoming data for the PR_NAME attribute into uppercase letters:

```
uUpper(IN.PR_NAME) ' OUT.PR_NAME
```

7. Enter **uUpper(IN.PR_NAME)** in the Transformation Rule column of the IN.PR_NAME attribute. The IN.PR_NAME value is forwarded to the OUT.PR_NAME attribute.
8. Click **Save** to confirm your settings. The green color of all ports in the project indicate that all components have been successfully configured.
9. Select **File** > **Save**.

Starting the Simulation

After you have added a data provider, data sink, and a data calculator, you are ready to start the simulation.

1. To initialize all components, click the **Start** icon on the toolbar.
2. Click **Step** to step through the project from component to component.

   At any point during the simulation, you can preview the current set of data, by right-clicking the link and selecting **Preview**. For example, when the first step executes, the data records are forwarded from the source component to the data calculator. A number on the link indicates the number of records transferred.

   Note: The Preview option is not available when there are no processed records or when no simulation data is available.

3. When all data has been processed, select one of:

   - **Execute post-processing as for successful execution** – to commit all tasks performed by the transactional components and to reset the project to its initial state.
   - **Execute post-processing as for failed execution** – to roll back all tasks performed by the transactional components and to reset the project to its initial state

   Click **Yes** to confirm resetting of the interactive trace. This clears all port buffers, releases temporary tables, and closes all database connections and temporary files.

If you click **No**, all open database connections and port buffers are retained. You can inspect, reconfigure, and then step through individual components again.

Using the Query Designer to Create Queries

Use the Query Designer to generate select statements inside queries, lookup definitions, preprocessing and postprocessing SQL statements.

Using the Query Designer you can:

- Browse the table catalog of any connected database of the current project
- Create SQL queries by using a graphical user interface
- Review generated SQL statements
- Execute SQL queries against the database
- Browse data in a selected table or view
- Create a table in the schema
- Delete all records in a table

  Note: To delete all records in a table, select **Enable delete functionality of database objects** in the **File** > **Preference** window.
- Count the number of records in a table or view

Opening the Query Designer

Use the Demo Getting Started project from the Demo Repository to create queries using the Query Designer.

1. In the Navigator, double-click the **Demo Getting Started** project to open it in the Design window.
2. Double-click the **DB Data Provider Full Load** component. Alternatively, select the component to display its properties in the Properties window.
3. Click the **Query** icon.
4. Click the **Query Designer** icon.

Creating a Simple Query

To generate a simple query that retrieves all attributes from a table, use the `PRODUCTS` table, available in the demo Getting Started project within the Demo Repository.

1. In the Navigator, select the **Model** tab, then click the table or view name. To search for a particular table or view name, press **Ctrl+F**.
2. Drag the selected object to the Design window.
3. To view the results of the generated query, select **View** > **Generated Query** or select the Generated Query tab.

Creating a Query Using Multiple Tables

To generate a query that joins and retrieves information from two tables, use the `PRODUCTS` and `SALES` tables.

1. Drag the **`PRODUCTS`** table from the Navigator to the Design window.
2. Drag the **`SALES`** table from the Navigator to the Design window.
3. Create a join between the tables by linking the `PR_ID` fields of both tables. To automatically create joins between identically named attributes within tables or views:
   a) Select **File > Preferences** from the main Sybase IQ InfoPrimer Development window.
   b) Select **Workbench > Query Designer**, then select **Create joins automatically**.
4. To view join attribute details, click the **Join** tab in the Query Designer window.

Modifying the Default Settings of a Join

A join between two tables is indicated by a line that connects the joining fields. The line is labeled with a join operator, which, by default, is called equijoin.

1. Right-click the line connecting the two joining fields.
2. Select **Modify.**
3. Select a join type.

Modifying the Sort Order of Joins

Edit the sort order of joins.

1. In the Join tab, right-click a row and select:
   - Move to start
   - Move up
   - Move down
   - Move to end
2. To revert to the default state of the join at any point, right-click a row and select Sort joins to default order.

Adding One or More Attributes to the Select Clause

Update the **select** clause to add one or more attributes.

1. Drag the **PRODUCTS** and **SALES** tables to the Design window, if they are not there already.
2. To add a single attribute, right-click the attribute you want to add, and select Add Items to Selection. To add more than one attribute, hold down the Ctrl key while you select the attributes you want to add.

Alternatively, click the **PRODUCTS** and **SALES** tabs in the Query Definition pane and select the attributes to add to the **select** clause. To search for an attribute, click the Search icon and provide your search criteria.

You can use an asterisk (*) as a wildcard to search for any number of unknown characters. For example:

- If your attribute is an integer datatype, your search criteria can be one of these:
  ```
  int, int*, i*ger
  ```
- If your attribute name contains “PROD” and has “CD” at the end, your search criteria can be:
  ```
  *PROD*CD
  ```

Adding All Attributes of a Selected Table to the Select Clause

Update the **select** clause to add all attributes of a selected table.

1. In the Query tab, click the header of the table.
2. Right-click and select **Add Items to Select**.

Viewing Attribute Details and Generated Query

See the query generated by the Query Designer along with the attribute details.

1. Select **View > Generated Query**, or select the **Generated Query** tab.
2. Click the appropriate tabs to view the attribute details.

Adding Functions to the Select Attribute

Update the **select** attribute to add functions.

1. In the **Select** tab, right-click the attribute to which you want to add functions.
2. Choose the functions to add.

Note: To enforce appropriate and reliable attribute names for viewing data and port structures, define alias names for all attributes to which functions have been applied.

See also

- *Use Column Aliases When Entering Queries* on page 525
- *Adding Alias Names to Selected Attributes* on page 190

Adding Alias Names to Selected Attributes

Include alias names for attributes.
In the **Select** tab, enter a name in the Alias column to enforce an output column name used in Data Viewer and associated port structure.

Using Templates to Create ETL Projects and Jobs

Use templates to automatically create projects and jobs.

Building a Migration Template Using the Template Assistant

The template assistant lets you create a new template or use an existing template to migrate data from one database to another.

1. Select **File > New > Template**. Alternatively, right-click **Templates** in the Navigator and select **New > Template**.
2. Enter the migration details:
   - Provide a name for the template. The name is used for the template object and, further qualified, for the generated transformation objects.
   - The migration type should be **DB to IQ**.
   - To use multiple engines for execution, select **Allow execution on multiple engines**.
   - To use multiple writers for loading data to the IQ database, select **Use IQ Multiplex**. Select this option to migrate more than one table to the IQ database.

     Note: To support multiplex execution, you must install the Sybase SQL Server 11 ODBC driver on the same machine as IQ InfoPrimer Development and IQ InfoPrimer Server.
   - To enable loading bulk load data into the IQ database from files located on remote host machines, select **Use IQ Client Side Load**.
   - To lock the target table in Exclusive mode and prevent it from being updated by concurrent transactions, select **Use IQ Lock Table**. If selected, no other transaction can execute queries or perform any updates against the locked table. Use IQ Lock Table also queues multiple projects that load the same table in Sybase IQ.

     If you select this option, you must also specify the maximum blocking time that the project should wait before acquiring the lock.
   - To enable transactionality for the generated job or projects, select **Transactional**. Data is committed at the end of the write operation for a successful execution, and rolled back for an unsuccessful execution.

     Note: If the Transactional option is selected, all the datasource and data sink components that support transactionality are created with their Transactional property enabled.
   - Click Next.
3. Enter connection details for the source database and select the tables to transfer.

   Note: The database connection properties are the same as for the DB components.

   Click **Logon** to view the list of available tables for the specified database. By default, each table is selected for transfer. Unselect the tables you do not want to transfer. You can also choose one or more table rows, right-click and select **Exclude**. To include a table for transfer, right-click and select **Transfer.**

Alternatively, you can click one of these icons:

- **Exclude all objects from transfer** to exclude all tables.
- **Include all objects in transfer** to include all tables.

To view additional information about a table, choose the table row, right-click and select:

- **Browse** – to view table data.
- **Count** – to view the record count of the selected table. To view the record count for all tables, click **Count All**.

Click **Next.**

4. Enter database connection properties for the destination database.

   Click **Logon** to view the list of available tables. To view the table data or the record count of the selected table, right-click and select **Browse** or **Count**. To view the record count of all tables, click **Count All**.

   Click **Next.**

5. Specify transfer settings for tables to be transferred.

   a) Select **Preserve schema/owner** to retain the schema or owner information of the source table.

   Note: The same schema or owner must exist in the destination database.

   b) Enter **stage** properties.

   In the Stage and Stage Server fields, specify the path for the load stage properties of the DB Bulk Load IQ component. If **Use Pipes** is selected, paths are automatically set. If Use Pipes is not selected, manually provide the values ended by the path delimiter. For example, `C:\ETLStage\`.

   Note: The Use Pipes option and the Stage server field are not available if you selected **Use IQ Client Side Load** in the migration details window in step 2.

   c) Select source attributes.

   By default, all attributes of a table are selected for transfer. To change the attribute selection, click the icon in the Columns field.

   In the **Select Attribute** window, unselect the attributes to exclude from transfer. You can also select one or more attribute rows, right-click, and choose **Exclude.**

   d) Select destination tables.

   Source and destination table names are generally the same. To use different names, enter a new name into the Destination field or select an existing table.

   e) Select additional options to perform appropriate actions for each table:

   - Data model options – before the transfer starts, verify that the destination tables exist. The data model options can help you set up the destination data model. They do not affect execution, but affect the data model when it is created from the template.

To create a new destination table based on the selected source attributes, select **Create Table**, or right-click the option and select **Activate**. To re-create an existing table, select **Drop Table**.

- **Execution** options – these options affect the execution on project level.

  Select **Truncate** to remove all records from the destination table before loading. This option corresponds to the `Truncate Table` property of the target component.

  The failure of a critical project causes the job to stop execution and signal failure. The Critical option and the Ignore Errors option correspond to the properties of the multiproject job component.

  The Ignore Errors setting does not affect the projects generated through this template.

f) Click Next.

6. Select the tasks to perform on the collected data.

Note: Except for saving the template, you can alternatively perform all tasks described here, by right-clicking a stored template in the Navigator.

- **Save template** – store the template in the repository. Storing allows you to reuse the collected data for similar jobs.
- **Build projects and jobs** – create one project for each source table, and a migration job that controls the execution of all the projects.
- **Create the destination data model** – set up the destination data model according to the data model options you entered. Click **Advanced** to enter SQL commands, which are executed before the destination tables are created.
- **Execute job** – available only if Build projects and jobs is selected. If you select this option, the generated job is executed after migration template data is processed.

Note: Select at least the Save template or Build Projects and jobs options to not lose collected data.

Click **Finish.**

Note: Before you can execute the generated job, either register engines or open the job and deactivate the MultiEngine Execution option.

While processing the data, you can view the current state and progress.

Creating a Template

Create a template using the template assistant.

1. In the Navigator, right-click **Templates.**

2. Select **New** > **Template**.

The template assistant guides you configuring a migration template.

Modifying a Template

Edit a template using the template assistant.

1. In the Navigator, either double-click the template, or right-click it and select **Open**.
2. Modify and save the template.

Copying a Template

Use the template assistant to copy a template.

1. In the Navigator, right-click the template.
2. Select **Copy**. Enter a name for the new template. You can also copy a template into a different repository.

Deleting a Template

Remove a template using the template assistant.

1. In the Navigator, right-click the template.
2. Select **Delete.**

Note: Deleting a template does not affect jobs and projects that are based on that template.

Renaming a Template

Rename a template using the template assistant.

1. In the Navigator, right-click the template and select **Rename.**
2. Enter a new name for the template.

Building a Job from a Template

Create a migration job and all related projects based on a stored template.
In the Navigator, right-click the template and select **Build**. To enforce unique names, a creation timestamp is added to all object names.

Creating a Data Model from a Template

Set up the destination data model according to the data model options stored with the template.
In the Navigator, right-click the template and select **Create Data Model**.

Managing Jobs

Use a job to set up powerful control flows for one or more projects. Within a job, you can run multiple projects sequentially or in parallel. Jobs control the order in which projects are executed. You can schedule to run a Sybase IQ InfoPrimer job without any user interaction.

A job that executes a single project consists of at least:

- A Start component
- A Project component
- A Finish component

You can extend a job to include multiple:

- Projects in sequential or parallel order
- Synchronizers
- Finish and error components

A Start component is always followed by one or more Project components.

Creating a Job

Use various job components to create a job.

1. In the Navigator, right-click **Jobs** and select **New** > **Job**.
2. Drag the **Start** component from the Component Store to the Design window.
3. Add the **Project** component and connect it to the **Start** component.
4. Add the **Finish** component and connect it to the Project component.
5. Double-click the **Project** component.
6. Select the project you want to include in this job. Click **Save.**

The job is now ready to be executed in Sybase IQ InfoPrimer Development or as a scheduled task.

From the Navigator, you can display and access the projects included in a job.

Modifying a Job

Edit an existing job.

1. In the Navigator, double-click the job name, or right-click the job, and select **Open.**

2. Modify the job and save the changes.

Copying a Job

Create a copy of an existing job, leaving the original untouched, and storing no references to the originating job.

1. Double-click the job you are copying to open it in the Design window.
2. In the Navigator, right-click the job and select **Save As**. Alternatively, select **File > Save As**.

   If you are working with multiple repositories, select the target repository.
3. Provide a name for the job. A copy of an existing job is created.

   Note: Copying a job to a different repository does not copy the projects included in the job. You must select projects from the new repository for all project and multiproject components in your job.

Deleting a Job

Remove a selected job and all its included projects.

1. In the Navigator, right-click the job.
2. Click **Delete**. By default, only the selected job is deleted. To delete the job and all included projects, select the **Delete Included Projects** option.

   Note: Before you delete the projects used in a job, make sure the projects are not used in other jobs. This is not checked automatically. Projects that are currently open for design and locked by any user are not affected.

Renaming a Job

Rename an existing job.

1. In the Navigator, right-click the job.
2. Select **Rename.**

Controlling Job Execution

You can control job execution in two ways.

- Using the Synchronizer component, which allows you to branch job execution based on a project's success or failure.
- Ignoring errors on each project.

Job Components

Learn about the various job components.

See also

- *Start* on page 197
- *Project* on page 197
- *Synchronizer* on page 198
- *Multi-Project* on page 199
- *Finish* on page 200
- *Error* on page 200

Start

Start is the first component you add to any job. To add this component to a job, drag it from the Component Store onto the Design window.

Note: You can connect multiple Project components, Multi-Project components, or both, to the Start component.

Project

The Project component identifies the project you want to run in the job. Use this component to run an individual project in a job.

Adding and Configuring the Project Component

Add and configure a project component to your job.

1. To add a Project component to a job, drag it from the Component store onto the Design window.
2. Connect the Project component with its adjacent components.
3. Double-click the component and select a project to execute. Alternatively, select **Project Name** in the Properties window to select a project.

Running the Project Component Demos

Run the sample jobs available in the demo repository for the Project component.

1. In the Navigator, click **Repository > TRANSFORMER.transformer.Repository > Jobs**.
2. Select:
   - **Demo Transfer all German Data**
   - **Demo Transfer U.S. Sales on an incremental basis**

Synchronizer

Synchronizer controls the flow of the job execution.

Use the Synchronizer component to control the flow of the job depending on the status of previously executed projects. You can define each project as critical or noncritical. Critical project failures cause the job flow to follow the branch at the Error port of the Synchronizer.

The Success and Error ports of the Synchronizer component can be connected to any of:

- Multiple Project components
- Multiple Multi-Project components
- Multiple Project and Multi-Project components
- Single Finish component or Error component

Adding and Configuring the Synchronizer Component

Add and configure a Synchronizer component to your job.

1. Add the component to the job and connect it with all the projects that signal execution status to this component.
2. (Optional) In the Property window:
   - Click the **Synchronize Options** icon to select critical projects.
   - Select **Commit Intermediate Work** if you want all tasks performed by the preceeding projects to be committed as soon as the projects have completed successfully.
   - Select **Propagate Rollback** to enable transactionality for the preceeding projects. If selected, data is committed at the end of the write operation for a successful execution, and rolled back for an unsuccessful execution.

Running the Synchronizer Component Demos

Run the sample jobs available in the demo repository for the Synchronizer component.

1. In the Navigator, click **Repository > TRANSFORMER.transformer.Repository > Jobs**.
2. Select **Demo Transfer all German Data**.

Multi-Project

The Multi-Project component provides a visual representation of the project groups within a job.

The Multi-Project component combines the properties of the Project and Synchronizer components. The Success and Error ports of the Multi-Project component can be connected to any of:

- Multiple Project components
- Multiple Multi-Project components
- Multiple Project and Multi-Project components
- Single Finish component or Error component

Use this component when your job consists of a large number of independent projects that can be executed in any order, and even in parallel, when used in multiengine jobs.

Adding and Configuring the Multi-Project Component

Add and configure a Multi-Project component to your job.

1. Add the component to the job and connect it to its adjacent component.
2. In the Properties window:
    - Click the **Projects Execution** icon to select the projects to execute.
    - (Optional) Select the **Commit Intermediate Work** icon if you want all tasks performed by the included projects to be committed as soon as the projects have completed successfully.
    - (Optional) Select **Propagate Rollback** to enable transactionality for the contained projects. If selected, data is committed at the end of the write operation for a successful execution, and rolled back for an unsuccessful execution.
3. To add projects to the group, right-click the Project name in the Navigator and select **Add Projects**.
4. To remove projects from the group, select the projects in the Navigator and and select **Remove Projects**.

Note: A project group can contain only one instance of a project. Do not add a project multiple times.

The option available for project execution is:

- **Critical** – define each single project as critical or noncritical. The failure of a critical project causes the Multi-Project component to signal failure.

Running the Multi-Project Component Demos

Run the sample jobs available in the demo repository for the Multi-Project component.

1. In the Navigator, click **Repository > TRANSFORMER.transformer.Repository > Jobs**.
2. Select Demo Transfer all U.S. Data.

Finish

Finish visually represents the end of a successful job execution. Use this component to mark the successful end of a job. You can connect the Finish component to these job components: Synchronize, Project, or Multi-Project.

Running the Finish Component Demos

Run the sample jobs available in the demo repository for the Finish component.

1. In the Navigator, click **Repository > TRANSFORMER.transformer.Repository > Jobs**.
2. Select:
   - **Demo Transfer all German Data**
   - **Demo Transfer all U.S. Data**
   - **Demo Transfer U.S. Sales on an incremental basis**

Error

The Error component visually represents the end of a failed job execution. Use it to mark the end of a failed job. You can connect the Error component to these job components: Synchronize, Project or Multi-Project.

Running the Error Component Demos

Run the sample jobs available in the demo repository for the Error component.

1. In the Navigator, click **Repository > TRANSFORMER.transformer.Repository > Jobs**.
2. Select:
   - **Demo Transfer all German Data**
   - **Demo Transfer all U.S. Data**

Execution

Includes information about projects and job execution.

Project Execution Using the Default Grid Engine

Execute projects in the default grid engine.

- For projects currently open in the Design window, select **Run > Execute**, or click the **Execute** icon on the toolbar.

  Execution impacts the state of the simulation. If you try to execute an unsaved project, you are prompted to save the project. If you save the project, all simulation data for the project is lost, and the project is executed.

  Note: Before execution, you must save project changes as the project definition is read from the repository. If you do not save the changes, execution does not start.
- In the Navigator, select the project to execute. Right-click and select **Execute Project**.

Before executing an EL Project, you can edit these runtime properties:

- Extract from source
- Load into target
- Row limit
- Restart failed project – allows you to restart execution from the point where the error occurred instead of executing the entire project from the beginning. For example, if an EL project that has 100 tables to be loaded fails after loading 75 tables due to a problem in loading the 76th table, you can fix the problem with the 76th table and restart execution to allow the remaining 25 tables to be loaded.

  Note: You can only restart a failed EL project. This feature is not available for SQL Transformation and ETL projects. To be able to restart a failed EL Project, you must set its transaction type property to Restartable at design time.

See also

- *Viewing Properties of the Executed Job or Project* on page 218

Job Execution Using the Default Grid Engine

You can execute a job directly from Sybase IQ InfoPrimer Development, or at specific time intervals as a scheduled task of the operating system task manager.

- To execute a job currently open in the Design window, select **Run** > **Execute**.
- To execute a job directly from the Navigator, right-click the job, and select **Job Execute**.
- To schedule a job, select **Tools** > **Runtime Manager**.

See also

- *Scheduling a Project and Job* on page 214

Executing Projects and Jobs Using IQ InfoPrimer Server

Sybase IQ InfoPrimer Server can perform execution of projects and jobs on all supported platforms, using the command line parameters.

To execute projects and jobs, use:

```
GridNode –-project PROJ-1234-5678 -–dbinterface dbodbc --dbhost
etl_comp
–-client transformer–-user TRANSFORMER –-password 1234ABCD
```

where project ID is “PROJ-1234-5678,” the database interface is “dbodbc,” host name is “etl_comp,” client is “transformer,” user is “TRANSFORMER,” and the encrypted version of the password is “1234ABCD”.

You can also use these command line parameters to execute projects and jobs:

- **project** – to specify projects and parameter sets by name. You can also specify jobs by name. Specifying names is easier than entering a complex project, job, or parameter set ID. This example uses the project name “LoadCustomers” and the parameter set name “myparams”:

  ```
  GridNode –-project LoadCustomers -–dbinterface dbodbc --dbhost
  etl_com
  –-client transformer –-user TRANSFORMER --paramset myparams –-
  password 1234ABCD
  ```
- **encrypt** – to generate an encrypted password. You must use encrypted passwords with **dbpassword** and **password**. To encrypt “mypassword”, enter:

  ```
  Gridnode --encrypt mypassword
  ```

  Sybase IQ InfoPrimer Server generates and displays the encrypted password.

- **ping** – to verify whether Sybase IQ InfoPrimer Server is running on a particular host and port. To verify whether Sybase IQ InfoPrimer Server is running on the default port on "localhost," enter:

  ```
  Gridnode --ping localhost
  ```

  If Sybase IQ InfoPrimer Server is running, you see:

  ```
  localhost is alive!
  ```

  You see an error message if Sybase IQ InfoPrimer Server is not running on the host and port you specify.
- **env** – to specify environment variables for projects and jobs. You can access the values for these variables at runtime with the **uGetEnv** function. This example uses the "LoadCustomers" project with the environment variables INPUT_FILE and OUTPUT_FILE:

  ```
  GridNode –-project LoadCustomers –-dbinterface dbodbc –-dbhost
  etl_com
  –-client transformer –-user TRANSFORMER --paramset myparams –-
  password
  1234ABCD --env "INPUT_FILE=input.txt;
  OUTPUT_FILE=output.txt"
  ```

Note: When you enter a command, type in the commands, parameters, and values in a continuous line. The examples in this section use multiple lines only for clarity.

In Sybase ETL versions earlier than 4.5, job and project execution was performed by the ProcessQ application. ProcessQ is now deprecated.

Command Line Parameters

Sybase IQ InfoPrimer Server command line parameters.

To display an overview of the available parameters, enter `GridNode --help`, or `GridNode -h` at the command prompt. The console output shows you the long and short forms for each parameter, for example:

```
--version,  -V    Displays version information
```

Note: The full parameter name is always prefixed by two minus signs, whereas the short form has only one.

Table 43. Sybase IQ InfoPrimer Server Command Line Parameters

Command	UNIX	Windows	Description
install or **inst**	No	Yes	Installs the application as a UNIX daemon or Windows service.
remove or **rm**	No	Yes	Removes system service start.

Command	UNIX	Win-dows	Description
setoptions or **so**	No	Yes	Sets the command line options to be used when running as a Windows service.
getoptions or **go**	No	Yes	Prints the command line options to be used when running as a Windows service.
background or **bg**	Yes	No	Sets background processes that do not overuse system resources.
no_pidfile or **nopid**	Yes	No	Sets the file to which the server records the daemon process ID.
console or **con**	Yes	Yes	Writes detailed error information and trace messages to the console.
diagnosis or **diag**	Yes	Yes	Lists application environment.
tracelevel or **tl**	Yes	Yes	Sets trace level for debugging from 0 (no trace) to 5 (very verbose).
server or **s**	Yes	Yes	Identifies the remote server to be used.
port or **p**	Yes	Yes	Identifies the port number to operate on.
version or **V**	Yes	Yes	Displays the application version information.
help or **h**	Yes	Yes	Displays help information.
licenses or **ll**	Yes	Yes	Identifies the short information about available licenses and their status.
nodelist or **nl**	Yes	Yes	Lists all known peer nodes.
shutdown or **sh**	Yes	Yes	Shuts down the node.
nodename or **n**	Yes	Yes	Sets a node name.
Command line parameters for executing projects and jobs			
dbhost *host*	Yes	Yes	Repository database host name or datasource name (DSN).
dbinterface *interface*	Yes	Yes	Repository database interface.
dbdatabase *database*	Yes	Yes	Repository database name.
dbschema *schema*	Yes	Yes	Repository database schema.
dbuser user	Yes	Yes	Repository database user.
dbpassword encrypted password	Yes	Yes	Repository database password.

Command	UNIX	Win-dows	Description
client *client*	Yes	Yes	Repository client name.
user *user*	Yes	Yes	Repository client user.
password encrypted password	Yes	Yes	Repository client password.
perflog [*args*]	Yes	Yes	Set performance log level to 0 or 1.
project [*name* | *ID*]	Yes	Yes	Execute a project by specifying the project name or ID.
job [*name* | *ID*]	Yes	Yes	Execute a job by specifying the project name or ID.
paramset [*name* | *ID*]	Yes	Yes	Specify a parameter set by name or ID for a project or job.
encrypt *password*	Yes	Yes	Encrypts a password, and displays the encrypted value. Use **encrypt** to generate the encrypted passwords you must use with **dbpassword** and **password**.
ping *host:port*	Yes	Yes	Checks if ETL Server is running at the host and port you specify.
env "*variable1=value*; *variable2=value*; ...; *variableN=value*"	Yes	Yes	Specify additional environment variables to run with Sybase IQ InfoPrimer Server. Use semicolons to separate multiple environment variables, and enclose the entire string of variables in double quotes.
repcdcinstancename or **ri**	Yes	Yes	Specify the Replication CDC Service Name.

Executing Multiple Projects Concurrently Using Sybase IQ InfoPrimer Server

To execute multiple projects concurrently on a single or multiple grid engines, or on the same remote grid engine, specify in the `Default.ini` file the maximum number of projects to execute concurrently.

Warning! Multiple users concurrently executing projects or jobs against the same remote engine may encounter problems when running projects that access the same resources such as database tables and files.

1. Navigate to the `etc` directory in the installation folder and use a text editor to open the `Default.ini` file.
2. In the Runtime section, set *MAXPROJECTS* to the number of projects to execute concurrently on a given grid engine.

   For example:

   ```
   MAXPROJECTS= 3
   ```

   If you set *MAXPROJECTS* to 0 or a negative number, there is no limit to the number of projects that can be concurrently executed on the grid engine.

   By default, the maximum number of projects that can be executed at one time is 10.

Note: The grid engine does not incorporate a locking mechanism for multiple users running projects against the same remote grid node. The source grid engine queries the target engine for the number of projects currently running and the configured maximum number of concurrently running projects. If the number of currently running projects is fewer than the maximum, the source engine executes the project on the target engine. If two source grid engines are executing projects or jobs simultaneously, the number of projects running on a grid engine may exceed the configured maximum.

INI File Settings

The `INI` files that include the settings for Sybase IQ InfoPrimer Server are in the `etc` folder of the installation directory.

Note: All communication between servers is done over TCP/IP on the default port for communication, 5124. You can change the port on all server installations to a different number in the `Default.ini` file, or by using the command line, if necessary.

Table 44. Default.ini File Setting Details

Group	Key	Values	Default	Description
Network	proxy	host:port explorer	explorer	Sets the proxy for Internet access. You can fine-tune the proxy for a certain protocol by using “http_proxy”, “https_proxy”, “ftp_proxy”, or “ftps_proxy”. The proxy value “explorer” takes the system proxy in Windows environments.
Network	timeout	1 – 2147483 seconds	600 seconds	Sets a timeout value for FTP connections.

Group	Key	Values	Default	Description
Language	Default	English_USA	English_USA	Tunes the behavior of the application based on the selected language and country.
Loader	ParallelLoads	ParallelLoads=n where, n is an integer greater or equal to zero	0	Loads tables in parallel by using parallel threads. By default the loader uses one loader thread for each cpu. On machines with large amount of cpus, you may want to limit the value.
Logging	Console	1/0	0	Sends log information to the console.
Logging	LogFile	1/0	1	Sends log information to the `system.log` file.
Logging	Tracelevel	0 – 5	0	Provides varying degrees of information for debugging. Level 0 provides the least information and level 5 provides the most detailed information. During normal execution, set this value to 0 to minimize performance impact.
Logging	Flushtime	1 – n	1	Indicates seconds between the internal log flashes.
SMTP	Server	*<name or IP address of SMTP server>*	<empty>	Sets the values for the SMTP server for alerting. The syntax for specifying a Server URL for the SMTP server is: `<protocol>://[user[:password]@]host[:port]` where: • *protocol* is SMTP or secure SMTP (SMTPs). • *user* is the SMTP username.
	Port	*<port of the SMTP listener>*	25	
	Sender	*<sender as it appears in the mail inbox>*	Sybase IQ InfoPrimer<noreply@ localhost>	
	Username	*<login name for SMTP server>*	<empty>	

Group	Key	Values	Default	Description
	Password	*<encrypted password for SMTP server>* **Note:** Encrypt the password using **GridNode --encrypt**.	<empty>	• *password* is the SMTP password. • *host* is the hostname or the IP address. • *port* is the listener port number. **Note:** Except for protocol, ServerURL can be substituted by using the single INI file keys, such as server, unsername, password, host, and port.
	Retry Count	*<positive integer describing how often ETL tries to send the mail>*	0	
	Retry Interval	*<positive integer describing the numbers of seconds between each retry>*	5	
	Server URL	*<default server URL>*	<none>	
	Recipients	*<comma separated list of default recipients>*	<none>	
	Subject	<your default e-mail subject>	<none>	
Runtime	Runtime	Keep_History_Days = 5MaxProjects = 10	Keep_History_Days = 5MaxProjects = 10	Indicates days after which entries in the execution history expires.
Path	Userdata	*<startup configuration file>*	`Userdata.conf`	Specifies the configuration file to be used when Sybase IQ InfoPrimer starts. Sybase IQ InfoPrimer includes two configuration files in the `./etc` directory, `userdata_user.conf` and `userdata_main.conf`. One of these files is copied to `userdata.conf` during installation.This parameter is set internally.

See also

- *uSMTP* on page 283

Reducing Job Execution Time Using Multiple Engines

The parallel grid architecture lets you run a job on multiple engines. A typical multiengine job contains multiple projects with few or no dependencies between them. Thus, projects can be executed on multiple engines at the same time after you have installed the multiple engines, and registered them for a special repository.

Sybase IQ InfoPrimer Server uses User Datagram Protocol (UDP) broadcasts to inform other servers about urgent events, such as starting and stopping, as well as system failure or crash. The default port for communication is 5124, which should not be in use or blocked by a firewall. All other communication between servers is done over TCP/IP on the same port.

Note: When executing a multiengine job from that repository, all projects are executed on those engines.

1. Double-click a job.
2. In the Design window, right-click and select **MultiEngine Execution**.

During execution, the job uses the registered engines to distribute the projects.

See also

- *Grid Engine Registration* on page 23

Executing Multiengine Jobs

Use the Navigator to execute multiengine jobs. Alternatively, use the Runtime Manager to schedule it.

1. in the Navigator, right-click the job.
2. Select **Job Execute**.

Parameter Sets

When you execute a project, all component properties are initialized with the values stored in the repository. Parameter sets let you overwrite some of these values.

For example, you can use parameter sets to change database connection settings when you move projects or jobs from development to production. To use parameter sets:

- Select the component properties to use as parameters.
- Store sets of parameter values.
- Assign a stored parameter set on execution.

Selecting Component Properties as Execution Parameters

Choose the component properties you want to use as execution parameters.

Note: For EL Projects, the Evaluate and Externalize properties must be set in the Advanced tab of the Configuration window. See Configuring an EL Project.

1. In the Properties window, right-click all component properties you want to use as execution parameters, and select **External.**
2. Right-click all component properties you want to assign a dynamic value using SBN expressions via a parameter set, and select **Evaluate**. Include all nonprinting values, such as Tab, CRLF, and so on.
3. Save the project.

Note: Provide unique names for at least all components that provide project parameters. You may also want to change the prompt and description of the properties.

Managing Parameter Sets

You can assign parameter sets to projects and jobs.

1. In the Design window or in the Navigator, right-click a project or job.
2. Select **Parameter Sets** to see a list of defined parameter sets for the selected project or job.

   Note: For some properties, the values displayed on project design may differ from the values that you must provide in a parameter set.

See also

- *Special Property Values* on page 212

Creating a Parameter Set

Learn how to create a parameter set.

1. In the Parameter Set window, click **Set > New**.
2. Overwrite the current values with the values to add.
3. Click **Save.**
4. Enter a name for the parameter set. Click **OK**.

Modifying a Parameter Set

Learn how to edit a parameter set.

1. In the Parameter Set window, choose a parameter set.
2. Click **Set** > **Open**.

3. Overwrite the current parameters with the new values.
4. Click **Save.**

Deleting a Parameter Set

Learn how to delete a parameter set.

1. In the Parameter Set window, choose a parameter set.
2. Click **Set > Delete**.

Copying a Parameter Set

Learn how to copy a parameter set.

1. In the Parameter Set window, choose a parameter set.
2. Click **Set > Copy**.
3. Name the new parameter set.
4. Click **OK.**

Executing a Project or Job with Parameter Sets

Learn how to execute a project or job with parameter sets.

1. In the Navigator, right-click a project or job.
2. Choose **Execute Project** or **Job Execute**.
3. Select an existing parameter set from the list, or click the **Add a Parameter Set** icon to open the parameter set window.
4. Create or modify a parameter set for use in the execution.
5. Click **Execute**.

Note: If no external properties are available in the project or job, the parameter set option does not appear in the Execution Monitor window.

See also

- *Viewing Properties of the Executed Job or Project* on page 218

Assigning Parameter Values

To select a single parameter, click the appropriate list row. To select multiple parameters, drag the mouse over the respective rows, or use Ctrl+click to select additional rows.

After you select a parameter:

- Enter the new value. The old value is overwritten. Or,
- Edit the existing value directly in the Value cell and click **Save.**

Assigning the Same Value to Multiple Properties

Because parameter sets are based on component properties, you may want to assign the same value to multiple properties.

1. Select the parameters to which to assign values.
2. Enter the new value and confirm that you want to use the value for all selected lines.

   Alternatively, click **Edit Selected Values** or right-click and select **Edit Selected**. Edit, then confirm, the value.

Sorting the Parameter List

You can use single or multiple columns to sort the parameter list.

Sorting Parameters by a Single Column

Sort parameters based on a single column by toggling between different sort orders.

1. Click the column header.
2. Click multiple times to toggle between ascending, descending, and the original sort order.

Sorting Parameters by Multiple Columns

Sort parameters based on multiple columns.

1. Click the primary column header multiple times to sort in the appropriate order.
2. Press **Ctrl** and click the secondary column header to sort the columns.

Special Property Values

There are some properties where values appear on project design differently from the values that you must provide in a parameter set.

Check Boxes

For properties represented by a check box on project design, enter 0 (deactivated) or 1 (activated) as the value in a parameter set.

Expressions

To use dynamic values in a parameter set, enter SBN expressions in the same way as in the Design window.

The Eval column indicates whether a property is enabled for expressions.

You especially need expressions when setting values containing nonprinting characters, such as, Tab, CRLF, and so on. You must set the Evaluate option for these properties when designing the project.

Note: You cannot validate expressions in the Parameter Set window.

See also

- *Square Bracket Notation* on page 178

Drop-down Menus

Some menus do not display the underlying parameter value. Some of these values require you to set the Evaluate option to assign them via a parameter set.

Use this table to identify which value (Value) corresponds to the shown one (Prompt) and whether you must enable Evaluate.

Component	Property	Prompt	Value	Evaluate
Database connections	Interface	ODBC	dbodbc	
		Sybase	dbsybase	
		Oracle	dboracle	
		IBM DB/2	dbdb2	
		OLE DB	dbole	
		SQLite Persistent	dbpersistent	
		INSERT LOCATION	dbinsertlocation	
Text formats	Row Delimiter	Position		
		LF	[uChr(10)]	x
		CR	[uChr(13)]	x
		CRLF	[uConcat(uChr(13),uChr(10))]	x
	Column Delimiter	Position		
		Tab	[uChr(9)]	x
		Comma	,	
		Semicolon	;	
		Pipe	|	
	Column Quote	None		
		Single Quote	'	
		Double Quote	"	

Component	Property	Prompt	Value	Evaluate
EL Source	Source Type (Source Type + File Format)	Database	1	
		Files: Delimited	2	
		Files: Fixed Length	3	
		Files: IQ Binary	4	
		Files: Large Object	6	
	Source Byte Order	Destination	1	
		Low (Little Endian)	2	
		High (Big Endian)	3	
EL Advanced	Stage File Format	Delimited Text	delimited	
		IQ Binary Data	binary	
	Transaction Type	Single Transaction	1	
		Auto Commit	2	
		Periodic Commit	3	
		Restartable	4	

Scheduling a Project and Job

Use the Runtime Manager to manage your scheduled tasks.

Note: You can schedule tasks only on grid engines running in the same subnet as IQ InfoPrimer Development. To schedule tasks on grid engines running in multiple subnets, run an instance of IQ InfoPrimer Development in the subnets in which these engines are running.

Invoke the Runtime Manager from the **Tools** menu and select the engine you want to use from the **Schedule services** list. A list of all tasks scheduled on that engine appear along with these details:

- Name
- Status
- Next Run
- Last Run
- Description

- Last Result – displays a value of either Failed or Succeeded for any scheduled task that has already run. The same information appears in the Execution Result column of the Schedule Task List when viewed in the Web Monitor.
  - Succeeded – either the task completed as expected or was prematurely terminated in one of these ways:
    - By clicking the **Terminate a Running Schedule** icon on the toolbar of the Runtime Manager, or by selecting **Actions > Terminate**.
    - By clicking **Cancel Execution** on the Execution Monitor.
    - By clicking **Cancel** in the Web Monitor.

    The value of Succeeded in the Last Result column does not always mean that the task ran to completion without errors.
  - Failed – means that the task was unexpectedly terminated, for example, by killing the grid engine process using the Windows Task Manager or through the operating system command line.

Note: Sybase recommends that you do not use the Windows Task Manager or operating system command line to terminate scheduled tasks.

Creating a New Schedule

Set up a new schedule using the Runtime Manager from the Tools menu.

1. In the Runtime Manager, select **Actions > Create**. Alternatively, click **Create a New Schedule** from the toolbar.
2. Select the project or job to execute. If required, select a parameter set or specify the Rep CDC Instance Name needed for incremental loading. Click **Next.**
3. Enter the schedule details:
   - Enter the schedule name, which must be unique.
   - Provide the start date and time for the schedule.
   - Click **Repeat Task** to specify how often you want the task to be repeated. If you are using the Scheduler to set up a repeated task using a server that is running on UNIX, you must configure the `.odbc.ini` file on the UNIX machine to include a pointer to the repository data server.
   - Click **Advanced** to specify:
     - **Execute new task concurrently** (default) – allows running multiple instances of the task concurrently.
     - **Execute new task sequentially** – waits for completion of the current process before executing the new one.
     - **Do not execute new task** – continues processing the current task and ignores the request to start a new task.
     - **Cancel the running task before executing new task** – stops the current process and starts the new one.

    - The **Stop the task after** option terminates a task that has been running for more than the specified duration. By default, this option is disabled.
  - Provide the end date, after which the schedule is inactivated.
  - Specify how often you want the task to be executed:
    - **Daily** – at a specified day interval.
    - **Weekly** – at a specified week interval.
    - **Monthly** – at a specified time on a particular day of each selected month. Specify the days of the month, and select the appropriate calendar months.
    - **Once** – only once at a specified date and time.
    - **At engine start-up** – each time the engine starts.

   Click **Next.**
4. Click **Finish** when you see a message that the schedule is successfully created.

The new schedule appears in the Runtime Manager along with the existing schedules, if any.

See also

- *Configuring .odbc.ini File* on page 218

Editing a Schedule

Use the Runtime Manager to modify an existing schedule.

1. Select the scheduled project or job to edit.
2. Click the **Edit** icon on the toolbar, or select **Actions > Edit**. Alternatively, you can double-click the scheduled project or job to modify.

Executing a Schedule

Run an existing schedule using the Runtime Manager.

1. Select the scheduled project or job to execute.
2. Click the **Execute** icon on the toolbar, or select **Actions > Execute**.

Note: A scheduled task uses the same performance logging level that has been set in the **Preference** window. To use a different log level, change the **Performance Logging** option before creating or editing the task, and then reset it after saving the task.

See also

- *Collecting Performance Data* on page 219

Deleting a Schedule

Remove an existing schedule using the Runtime Manager.

1. Select the scheduled project or job to delete.
2. Click the **Delete** icon on the toolbar, or select **Actions > Delete**.

Setting Refresh Options for Schedules

Enable the **Auto-Refresh** option in the Runtime Manager to periodically update schedule information.

1. Select **Actions > Enable Auto-Refresh**. By default, this option is unselected.

   Note: If **Auto-Refresh** is not enabled, the information you see may be old. Manually refresh the schedule to view the current status of tasks.

   Select **Action > Refresh Interval** to define the intervals at which to update the schedules. The refresh interval value should not be less than 2 seconds.
2. Click **OK.**

Importing Tasks to IQ InfoPrimer Scheduler

Import tasks created on the Windows Scheduler to IQ InfoPrimer Scheduler.

Note: You can only import scheduled tasks created in Sybase ETL 4.8 or earlier.

1. From the **Schedule Services** list on the toolbar, select the target engine to which to import the Windows scheduled tasks.
2. Select **Actions > Import**.
3. Select the source engine from which to import the scheduled tasks and click **OK**. Windows scheduled tasks on the selected engine are imported to the IQ InfoPrimer Scheduler.
4. Tasks are imported using the original name unless a task with the same name exists on the IQ InfoPrimer Scheduler. If the current name already exists, you must provide a new name.

   Note: The name of the scheduled task must be unique.
5. Click **Yes** to delete the Windows schedule.

   Click **No** to import the tasks from the source engine to any other engine before deleting them, or to manually delete the tasks later, using the Windows Task Scheduler.
6. You see a message showing the result of the import. Click **OK.**

Terminating a Schedule

Use the Runtime Manager to stop a schedule.

1. Select the scheduled project or job to terminate.

2. Click the **Terminate** icon on the toolbar, or select **Actions** > **Terminate**.

Configuring .odbc.ini File

Configure the `.odbc.ini` file on the UNIX machine to include a pointer to the repository data server before setting up a repeated schedule task.

1. Use a text editor to open the `etc/.odbc.ini` file.
2. Add an entry similar to:

   ```
   [repository_data_server]
   uid=dba
   pwd=sql
   EngineName=demo
   CommLinks=tcpip(host=<hostname>;port=<portnumber>)
   AutoStop=no
   DatabaseName=demo
   ```

   where *repository_data_server* is the name of the SQL Anywhere data server.

   Note: Make sure that the entry does not include the "AutoPreCommit" connection parameter.
3. Save the file.
4. In the Scheduler, set up the repeated task, then run the job.

Viewing Properties of the Executed Job or Project

The Execution Monitor displays the properties of the current job or project.

1. Select the project to execute.
2. Click **Execute** on the toolbar.

   If you have defined external properties for your project (EL and SQL Transformation projects), you also have the option to select the parameter set and the runtime properties.

The Execution Monitor window is divided into three panes, Job, Projects and Events. The top part of the Execution Monitor window displays properties of the currently executing job. The Projects list contains a line for every project in the job, and the Event list displays the events and alert details.

Saving and Copying the Execution Results

Save and copy the results of the project you are executing to an HTML file.

1. To save the results of the project, click **Save Results**.

2. To copy and paste the execution results of the job or project to a different application, such as Notepad or Microsoft Word, right-click the job or project in the Execution Monitor, and select **Copy.**

Cancelling Job Execution

Terminate the execution of a job.
To cancel job execution, click **Cancel Execution** in the Execution Monitor window.
Grid engines try to cancel running projects. Projects waiting to be executed are not started.

Using Content Explorer to Browse Schema Information

Browse schema information and data content of all connected datasources using the Content Explorer.

You can also generate ad hoc queries, which cannot be saved to a file or to the repository. To save generated SQL statements, select and copy the generated query from the Generated Query window.

Use one of these methods to open the Content Explorer:

- Select **Tools > Content Explorer**. The Choose Data Source window displays all the components currently connected to datasources. The names in the list of currently connected databases is a combination of a user-defined name and the generic name of the component type. Select a component and click **Start** to open the Content Explorer.
- Right-click a database component and select **Content Explorer**.

Analyzing Performance Data

While executing jobs and projects, Sybase IQ InfoPrimer collects data and stores it in a repository table.

Collecting Performance Data

Gather and store performance relevant data to a repository.

1. Select **File > Preferences > Performance Logging**.
2. From the **Logging Level** list, select 1.

Viewing Performance Data

Use the Navigator to view performance details.

Right-click a project or job and select **Performance Data**. Alternatively, open the project, right-click anywhere in the Design window, and select **Performance Data**.

The Performance Data window appears, displaying the execution details for the selected project or job under an **Overview** tab. By default, the performance data of executions performed within the last month appear. To change the range of execution dates that are shown, change the values of the **Execution Date Range** on the toolbar.

Note: You can view performance data of one project or job at a time.

To permanently remove performance data of any execution, select the specific row and click the **Delete selected performance log entries** icon on the toolbar. Alternatively, press **Ctrl +D**.

Warning! You cannot recover deleted performance data.

Viewing Project Performance Data

Use the Overview tab to view project performance details.

Select a row in the **Overview** tab and click the **Drill down in performance data** icon on the toolbar. Alternatively, double-click the selected row or the corresponding bar in the chart displayed. A new tab displays the details.

Note: When you view the performance data of a project using the IQ LoaderDB via Insert Location component or the IQ Loader File via Load Table component, the performance table may not display the correct value for the number of records read and written. IQ loads data directly from files or remote databases; therefore, this information is not available to Sybase IQ InfoPrimer. Information is available in the IQ Message log.

To view performance details for a selected component, select a row in the tab displaying the component details, and click the **Drill down in performance data** icon on the toolbar. Alternatively, double-click the selected row or the corresponding bar in the chart. The component performance details such as event name, duration, records read, and records written appear.

To return to the higher level of performance details, click the **Drill up in performance data** icon on the toolbar. To go back to the **Overview** tab, click the **Return to performance data overview** icon on the toolbar, or select **Navigation > Return to Overview**.

View Job Performance Data

Use the Overview tab to view job performance data.

Select a row in the **Overview** tab and click the **Drill Down in Performance Data** icon on the toolbar. Alternatively, double-click the selected row or the corresponding bar in the chart. A new tab displays details, such as project name, duration, records read, records written, result, and load time.

To view performance details for a selected project, select a row, and click the **Drill Down in Performance Data** icon on the toolbar. Alternatively, double-click the selected row or the corresponding bar in the chart. The project performance details appear.

To return to the higher level of performance details, click the **Drill Up in Performance Data** icon on the toolbar. To return the **Overview** tab, click the **Return to Performance Data Overview** icon on the toolbar, or select **Navigation > Return to Overview**.

Note: To find the selected component in a project, select **Tools > Show Component**. Alternatively, click the **Show Selected Component** icon on the toolbar.

Searching for Performance Data

Locate specific performance data from the Performance Data window.

1. Select any row in the performance data table.
2. Press **Ctrl+F** to open the search window. Enter your search criteria in the **Find** field.

Printing Performance Data

Generate a report for your performance data.

1. Select **Tools > Generate Report**.
2. Select the level of details that you want to print and choose the destination file. Available level of details include:
   - Overview
   - Job Performance (appears only when you are printing performance data reports for jobs)
   - Project Performance
   - Component Performance
3. Enter a destination file path for the generated report, or accept the default value.
4. To limit the report data to the executions selected in the **Overview** tab, select **Only Selected Executions**.

   Note: Only Selected Executions is available only if you select an execution in the Overview tab.
5. Click **Generate**, then click **Yes** to view the performance data report.

The report displays tabular and graphical data. It includes a table of contents that provides links to corresponding sections within the report.

The Performance Overview section of the report displays the duration of each project or job execution. The Project, Component, and the Job performance sections display a table and a chart of performance data for the selected execution, each showing a different level of data granularity.

Performance Data Model and Content

Performance data is stored in a single, denormalized, repository table named `TRON_PERFORMANCE`, which you can query to collect performance data.

Each execution of project or job is identified by a global unique ID. Additional information is provided about the account that initiated the execution and the repository in which the project or job is located.

Note: The starting time for execution and events are logged in Coordinated Universal Time (UTC), also referred to as Greenwich Mean Time (GMT).

For each executed project ID, version (modification date), name, and a global, unique execution ID is reported. A single project execution event stores the duration of a project in ms.

Project components are represented by ID, name, class, type, and version. The process event provides the number of steps and the amount of processed records.

For port events, the ID, name, class, type and the amount of input or output blocks and records are reported.

For each job, the ID, the version (modification date), and the name is reported. A single job execution event stores the duration of a job in ms. The components (projects) of a job are represented by their ID, class, and version.

Events

The performance log is based on events. For each event, the starting time and the duration (in ms) is stored. The description of an event is made up by a class, a name, and a text. Some events include a result (such as succeeded or failed). Also included is information about the engine that reported an event.

Table 45. Reported Events

Class	Name	Description
control	execute job	Total execution time of a job (one record per job execution, job duration in ms, in attribute PRF_JOB_DURATION).
init	load job	Get job definition from repository.
control	execute project	Total execution time of a project (one record per project execution, project duration in ms, in attribute PRF_PRJ_DURATION).
init	load project	Get project definition from repository.
init	create	Create project and component instances.

Class	Name	Description
init	configure	Configure project and component instances.
perform	prepare	Perform component preprocessing.
perform	process	Perform component step.
perform	finish	Perform component postprocessing.
perform	read	Get data to component input port.
perform	write	Push data from component output port.

Note: Due to distributed multithreading, the total project execution time can be significantly shorter than the sum of the execution time of all participating components.

Monitoring Projects and Jobs

Use a Web browser to monitor projects and jobs.

Prerequisites

- Start the Sybase IQ InfoPrimer Server, if it is not already running.
- Verify that you have Internet Explorer (IE) 6.0 or later installed on your machine.
- In the Internet Explorer, go to **Tools > Internet Options > Security** and make sure the security setting is set to Medium. Job information may not be displayed correctly if the security setting is set to Medium-high or above.

Task

1. Open the Web browser.
2. Enter:

   ```
   http://<hostname>:<port_number>
   ```

   where *<hostname>* is the network name of the machine on which the Sybase IQ InfoPrimer Server is running, and *<port_number>* is the port on which the node was started. The default port number is 5124.
3. To view a list of the running jobs, click the Active Jobs tab.

   You see all the running jobs along with details such as name, status, number of projects within the job, start and end time, and the number of records processed.
4. To suspend an active project and change its status, click the **Suspend** icon next to the project in the Active Jobs tab.

   To suspend a job and all projects included within it, click the **Suspend** icon next to the job in the Active Jobs tab or the Node Summary tab. The status of the job changes to Suspended. All projects in the job are also suspended.
5. To restart a suspended project and change its status, select the job that has a suspended project in the Active Jobs tab and click the **Resume** icon. The projects status changes to Running and the project is resumed.

   To restart a suspended job and all projects within it and change their status, click the **Resume** icon next to the suspended job in the Active Jobs tab.
6. To view job history, select the job in the Active Jobs tab or the Node Summary tab. Alternatively, click the Job History tab.
7. To cancel a project and change its status, select a job from the list of active jobs in the Active Jobs tab and click the **Cancel** icon. The project status changes to Cancelled and the project is stopped.

To terminate an active job, click the **Cancel** icon next to the job in the **Summary** tab. The job is removed from the list.

8. Select the Node Overview tab to review the details of all servers that are currently running such as, status, number of jobs, amount of CPU, memory, and disk space utilized by the server.
9. To stop and shut down a selected server, click the **Stop** icon next to the server name in the Node Overview tab. The server is removed from the list.
10. Select the Log History tab to review the system log details such as timestamp, type of error, and the error message. Click **Download** to download the log file on to your machine.
11. Select a grid engine from the list shown on the left side of the page to view the list of tasks scheduled on it. Click **Schedule Task List** from the Node menu. A list of scheduled tasks available on the selected engine appear. Click the **Terminate** icon next to the task to stop it. Similarly, click the **Start** icon next to a terminated task to execute it.

    Note: Tasks scheduled on Windows Scheduler are not shown.
12. Click **Alert History** from the Node menu to review the details of an alert. Click **Download** to download the alert log file on to your machine.

ETL projects for Slowly Changing Dimensions

Common SCD scenarios and how these scenarios are implemented using ETL projects and jobs.

Overview

Slowly changing dimension is a common data warehousing scenario. SCD utilizes three different method types for handling changes to columns in a data warehouse dimension table.

Type 1
In Type 1, new data overwrites existing data. The existing data is lost and there is no tracking of historical changes. Type 1, is the easiest method to support, but is useful to track historical changes.

Consider a table that keeps product information:

Key	Name	Price
1	Notebook	1200

The price of the notebook increases to 1500. The updated table overwrites the current record:

Key	Name	Price
1	Notebook	1500

Type 2
Type 2 retains the full history of values. If new data differs from the old data, an additional dimension record is created with new data values and becomes the current record. Each record contains the effective date and an expiration date to identify the time period for which the record was active. Use type 2 to keep a full history of dimension data in the table.

Type 3
Type 3 tracks changes using separate columns. There is one version of the dimension record that stores the previous value and current value of selected attributes. Use type 3 to track historical changes that occur only for a finite amount of time.

Consider a table that keeps product information:

Key	Name	Price
1	Notebook	1200

The price of the notebook increases to 1500 on 15th July 2008. To accommodate type 3, new columns are added, Current Price and Effective Date:

Key	Name	Original price	Current price	Effective date
1	Notebook	1200	1500	2008-07-15

Note: Type 3 is rarely used because altering the structure of the dimension table should be undertaken for only a very important change.

See also

- *Case Study Scenario* on page 228

Case Study Scenario

A case study scenario for type 2 SCD and information on how to create transformation projects in ETL to implement this scenario.

Case Description
You have two tables:

- `PRODUCT` in the operational or source database.
- `PRODUCT_PRICE` in the data warehouse or target database. This table tracks modification of products in the source table (`PRODUCT`) over time, such as:
  - Change in price of existing products
  - Newly added products
  - Deleted products

Table 46. Database Table Schema Of The `PRODUCT` Table

Column	Description
Key	Unique ID of product
Name	Name of the product
Price	Price of the product

Table 47. Database Table Schema Of The `PRODUCT_PRICE` Table

Column	Description
Key	Source key identifier in the source table (`PRODUCT`)
Name	Name of the product

Column	Description
Price	Price of the product
Valid From	Date of insertion of new records
Valid To	End of validity of records. A record becomes invalid when a new record with the same source key is inserted in the `PRODUCT_PRICE` table.

Rules

The rules to transfer dimensions from the `PRODUCT` table to the `PRODUCT_PRICE` table are:

1. If the record does not exist in the `PRODUCT_PRICE` table, create it with the same column values as the `PRODUCT` table. Set the Valid From date to the record insertion date, and the Valid To date to 9999-12-31.
2. If the record exists in the `PRODUCT_PRICE` table, but nothing has changed, do not insert a new record or update an existing one.
3. If the record exists in the `PRODUCT_PRICE` table, and the price of the record has changed:
   - Set the Valid To date for the record with the old price to yesterday.
   - Create a new record. Set the Valid From date to the record insertion date, and the Valid To date to 9999-12-31.
4. If the record exists in the `PRODUCT_PRICE` table but has been deleted from the `PRODUCT` table, set the Valid To date of the product in the `PRODUCT_PRICE` table to yesterday.

Note: A running history of dimension changes, based on these rules, is maintained in the `PRODUCT_PRICE` table.

How It Works

After initial load on 01 January 2008, the `PRODUCT` table appears:

Key	Name	Price
1	Notebook	1200
2	Monitor	1000
3	Mouse	500

After the application process is run for the first time, the `PRODUCT_PRICE` table appears:

Key	Name	Price	Valid from	Valid to
1	Notebook	1200	2008-01-01	9999-12-31
2	Monitor	1000	2008-01-01	9999-12-31
3	Mouse	500	2008-01-01	9999-12-31

On 15 January 2008, the `PRODUCT` table is updated when the price of the monitor is modified.

Key	Name	Price
1	Notebook	1200
2	Monitor	1400
3	Mouse	500

After the application process is run again, the `PRODUCT_PRICE` table appears:

Key	Name	Price	Valid from	Valid to
1	Notebook	1200	2008-01-01	9999-12-31
2	Monitor	1000	2008-01-01	2008-01-14
3	Mouse	500	2008-01-01	9999-12-31
2	Monitor	1400	2008-01-15	9999-12-31

On 22 January 2008, the `PRODUCT` table is updated again when a new product, a hard disk, is added.

Key	Name	Price
1	Notebook	1200
2	Monitor	1400
3	Mouse	500
4	Hard Disk	1000

After the application process is run again, the `PRODUCT_PRICE` table appears:

Key	Name	Price	Valid from	Valid to
1	Notebook	1200	2008-01-01	9999-12-31
2	Monitor	1000	2008-01-01	2008-01-14

Key	Name	Price	Valid from	Valid to
3	Mouse	500	2008-01-01	9999-12-31
2	Monitor	1400	2008-01-15	9999-12-31
4	Hard Disk	1000	2008-01-22	9999-12-31

The `PRODUCT` table is updated again on 28 July 2008, to remove the mouse as an available product.

Key	Name	Price
1	Notebook	1200
2	Monitor	1400
4	Hard Disk	1000

After the application process is run again, the `PRODUCT_PRICE` table appears:

Key	Name	Price	Valid from	Valid to
1	Notebook	1200	2008-01-01	9999-12-31
2	Monitor	1000	2008-01-01	2008-01-14
3	Mouse	500	2008-01-01	2008-07-27
2	Monitor	1400	2008-01-15	9999-12-31
4	Hard Disk	1000	2008-01-22	9999-12-31

Setting up ETL projects for SCD

Review the ETL concepts for accomplishing type 2 SCD using projects and jobs. The demo repository that is packaged with the product includes various transformation objects related to the type 2 use case, including:

Projects

- Demo Product Price SCD – Initial Load

  This project initializes or reinitializes the demo environment for the SCD – Update projects and job.

  Note: This project is not a part of the use case implementation. In a production environment, the first execution of the Update New and Modified project, on an empty target table, performs the initial load where all source records are processed as new records. Since the demo environment uses two different tables to simulate changes on the source data, the original data always needs to be restored in the target table by executing this project before running the other 2 update projects or the job.

- Demo Product Price SCD – Update New and Modified

  This project updates the dimension table of the target database on a daily basis to reflect modification or addition of products in the source database. See Rules 1 – 3 in Case study scenario.

  To accomplish a full update, also execute the SCD – Update Deleted project.
- Demo Product Price SCD – Update Deleted

  This project updates the dimension table of the target database on a daily basis to reflect deletion of products in the source database. See Rule 4 in Case study scenario.

  To accomplish a full update, also execute the SCD – Update New and Modified project.

Job

- Demo Product Price SCD – Daily Update

  This job executes both the SCD – Update New and Modified and SCD – Update Deleted projects, and provides a single transformation object for performing a full update of the target dimension table. Before executing this job, execute the SCD – Initial Load project.

See also

- *Case Study Scenario* on page 228

Understanding Target Dimension Table

Learn how to identify current target records.

Identifying Target Records

A target dimension table contains multiple records for the same source key. To differentiate a current version of a record from a historical version, the target dimension table uses a compound key, which includes the source key and either the effective date or the expiration date attribute. The ETL demo projects use the Valid From date attribute as part of the key.

Current Target Records

Each record in the source table is represented by exactly one current record in the target table. Only current records are relevant for SCD when checking the target dimension table. In the ETL demo projects the current records have a Valid To date of 9999-12-31.

Detecting Source Changes

Learn how to capture changes in the source table, including new source records, modified source records, and records that have been deleted from the source. Methods can be combined to detect different types of data changes in one step.

In the case study scenario, the source database does not contain any change log information, so source and the target content must be compared to detect any changes. Since, in most cases, the source and target objects do not reside in the same database, a heterogeneous join needs to be performed.

Detecting New Source Records

Records that are added to the source after the last update of the target dimension table do not have a corresponding current record in the target dimension table.

1. All source records are read using an appropriate Data Provider component.

   Note: All attributes that are transferred to the target dimension table are read, although only the key attribute is required for detection.

2. The existence of a corresponding current record in the target dimension is checked for each source record based on the source key attribute.
   1. Choose an appropriate Lookup component.

      To perform calculations on the data to be transferred, consider using the lookup functionality of the Data Calculator component.
   2. Select the lookup data from target. As this is a simple existence check, only the original source keys are needed from all current target records. However, lookups always return a value for a key, so you must also select an appropriate return value.
   3. Add an attribute to the port structure to populate the lookup result. The lookup result determines whether a source record is newly added or existed previously. This attribute indicates the data state and allows data to be filtered in the next step of the transformation process. See Modifying Port Structures. The new attribute is selected as the value attribute of a Lookup component or the output attribute in the Data Calculator.
   4. Set an appropriate lookup default value. The default value is returned by the lookup for nonexisting keys. To ensure that the lookup value correctly indicates the existence of records, set it to a constant that is different from all lookup values for any existing keys.

      Example:
      - Source data – **select Key, Name, Price from PRODUCT**
      - Lookup data – **select Key, '1' from PRODUCT_PRICE where Valid_To = '9999-12-31'**
      - Value attribute – Exists (integer)
      - Default value – 0
      - Performing this lookup results in records with attributes Key, Name, Price, Exists. The value of the Exists attribute will be 1(lookup value) for all records existing in target, and 0 (default value) for all nonexisting records.

Detecting Modified Source Records

Records that are modified in the source after the last update of the target dimension table have a corresponding current record in the target dimension table, but the relevant values are changed.

1. All source records are read using an appropriate Data Provider component.

   Note: All attributes that are transferred to the target dimension table are read, although only the key attribute is required for detection.

2. The existence of a corresponding current record in the target dimension table is checked for each source record based on the source key attribute, and the values are compared.
   1. Choose an appropriate Lookup component.

      Use the Data Calculator component to look up multiple values for a single key attribute and perform comparisons.
   2. Select the lookup data from the target. The key attributes and all values to be compared for all current target records are read using an appropriate Data Provider component.
   3. Add an attribute to the port structure to populate the additional target key attribute. The lookup result determines whether a source record has been modified or is either new or unchanged. This attribute indicates the data state and allows data to be filtered in the next step of the transformation process. The update operation on the target must uniquely identify the current record, thus the date part of the target key needs to be populated as well. See Modifying Port Structures.
   4. Set an appropriate lookup default value. The default value is returned by the lookup for nonexisting keys. To ensure that the lookup value correctly indicates the existence of records, set it to a constant that is different from all lookup values for the existing keys.
   5. Look up all necessary target values. The first lookup uses the new target key attribute as the output attribute in the Data Calculator, thus indicating existence. The values to be compared are read into temporary variables.
   6. Compare source and target attribute values. The target key attribute is recalculated, based on a value comparison for the existing records.

      Example:
      - Source data – **select Key, Name, Price from PRODUCT**
      - Lookup data – **select Key, Valid_From, Price from PRODUCT_PRICE where Valid_To='9999-12-31'**

      First check existence by reading the effective date from target:
      - Output attribute – Valid_From
      - Default value – 0

      Read current target price into temporary variable:
      - Output Attribute – Tmp_Price
      - Default Value – 0

      If a current target record exists (Valid_From is not 0), compare Price and Tmp_Price. Recalculate Valid_From to 0 if the Price has not changed. Performing these lookups and calculations results in records with Key, Name, Price, and Valid_From attributes. Valid_From either contains the effective date of the target record to be updated or contains 0, indicating new and unchanged records.

Detecting Deleted Source Records

Records that have been deleted from the source after the last update of the target dimension table still have a corresponding current record in the target dimension table.

1. Key attributes of all current records in the target dimension table are read using an appropriate Data Provider component.
2. The existence of a corresponding record in the source is checked for each current target record based on the source key attribute.
    1. Choose an appropriate Lookup component. See Lookup components.

       If your source data does not reside in a database, use the lookup functionality of the Data Calculator component.
    2. Select the lookup data from source. As this is a simple existence check, only the source keys are needed from all source records. However, lookups always return a value for a key, so you must select an appropriate return value as well.
    3. Add an attribute to the port structure to populate the lookup result. The lookup result determines whether a source record has been deleted. This attribute indicates the data state and allows the data to be filtered in the next step of the transformation process. The new attribute is selected as the value attribute of a Lookup component or the output attribute of a Data Calculator rule. See Modifying Port Structures.
    4. Set an appropriate lookup default value. The default value is returned by the lookup for nonexisting keys. To ensure that the lookup value correctly indicates the existence of records, set it to a constant that is different from all lookup values for the existing keys.

       Example:
       - Target data – **select Key, Valid_From from PRODUCT_PRICE where Valid_To='9999-12-31'**
       - Lookup data – **select Key, '0' from PRODUCT**
       - Value attribute – Removed (integer)
       - Default value – 1

       Performing this lookup results in records with attributes Key, Valid_From, Removed. The value of the Removed attribute will be 0 (lookup value) for all existing source records and 1 (default value) for all nonexisting records.

Alternatives

- If the source is a database that provides ascending indicator for insertions, updates, or deletions (like autoincrements, modification dates, and so forth), the DB Data Provider Index Load component can be used to read records changed since the last load only.
- Use a Staging component to load relevant data from both the source and target to the same database. New, modified, and deleted records are then detected by extracting data from the stage using a full outer join.

See also

- *Data Calculator JavaScript* on page 106

- *DB Data Provider Index Load* on page 75
- *DB Staging* on page 120
- *Source Components* on page 72
- *Lookup Components* on page 115
- *Modifying Port Structures* on page 69

Filtering the Records

Learn how to filter the records.

Use the Data Splitter component to remove records from the data stream, apart from splitting data streams to more than one output. See “Data Splitter JavaScript.” To remove records from the data stream, define conditions for every OUT-port, such that the records to be removed do not match any of them. To output a single data stream, configure a Data Splitter with a single output port by deleting one of the default output ports.

See also

- *Data Splitter JavaScript* on page 110

Populating the Target Dimension Table

Learn how to assign values to target attributes and perform partial updates.

Assigning Values to Target Attributes

Using the insert and update options of the DB Data Sink components, values are assigned to those target attributes that are not included in the inbound data stream. The values are constant for all records processed during a single execution but allow for SBN expressions to dynamically initialize them. In the ETL demo projects and jobs, the insert options use the values:

- Dynamic – **'[uDate('now','localtime')]'** (today) for the Valid From date attribute.
- Static – '9999-12-31' for the Valid To date attribute.

The update options use the dynamic value **'[uDate('now','localtime','-1 day')]'** (yesterday) for the Valid To date attribute.

Performing Partial Updates

The update options of a DB Data Sink component allow a subset of attributes to be updated, instead of updating the complete record. To exclude attributes from update, unselect them in the update options window. In the ETL demo projects, old records are outdated by updating the Valid To date attribute.

See also

- *DB Data Sink Update* on page 146
- *DB Data Sink Insert* on page 141

Function Reference

All functions supported by Sybase IQ InfoPrimer.

Supported Functions for ETL Project

List of supported functions that can be used in an ETL project.

Note: Although these functions are used mostly in ETL projects, the EL and SQL-T projects support some use of SBN expressions in which these functions may be used.

Aggregation

Review the list of Aggregation functions.

uAvg

Returns the average value over all input values

Syntax

```
real uAvg(value, ...)
```

Parameters

- **number value –** A list of numeric arguments

Examples

- **Example 1 –**

  ```
  uAvg(1,2,3,4,5) // returns 3
  ```

uMax

Returns the maximum from a list of values

Syntax

```
uMax(value,...)
```

Parameters

- **number value –** A list of numeric arguments

Examples

- **Example 1 –**

  ```
  uMax(1, 6, 4, -6) // returns 6
  ```

  ```
  uMax("b", "A", "a") // returns "b"
  ```

  ```
  uMax("2004-05_02", "2006-12-12", "1999-05-30") //
  returns "2006-12-12"
  ```

uMin

Returns the minimum from a list of values

Syntax

```
uMin(value, ...|)
```

Parameters

- **number value –** A list of numeric arguments

Examples

- **Example 1 –**

  ```
  uMin(1, 6, 4, -6) // returns -6
  ```

  ```
  uMin("b", "A", "a") // returns "A"
  ```

  ```
  uMin("2004-05-02", "2006-12-12", "1999-05-30")
  //returns "1999-05-30"
  ```

Bit Operations

Review the list of Bit Operation functions.

uBitAnd

Bitwise AND operation

Syntax

```
number uBitAnd(value, ...)
```

Parameters

- **number value –** A list of numeric arguments

Examples

- **Example 1 –**

  ```
  uBitAnd(10, 3) // returns "2"
  ```

uBitOr
Bitwise OR operation

Syntax
```
number uBitOr(value, ...)
```

Parameters

- **number value –** A list of numeric arguments

Examples

- **Example 1 –**
  ```
  uBitOr(10, 3) // returns "11"
  ```

Boolean
Review the list of Boolean functions.

uIsAscending
Returns **1** if every parameter is equal to or greater than its predecessor

Syntax
```
number uIsAscending(params, ...)
```

Parameters

- **params –** A list of expressions or values of any datatype.

Examples

- **Example 1 –** Check multiple values for an ascending order:
  ```
  uIsAscending("A", "B", "C") // returns 1

  uIsAscending("A", "A", "C") // returns 1

  uIsAscending("A", "C", "B") // returns 0

  uIsAscending("1", "2", "3") // returns 1

  uIsAscending("3", "2", "2") // returns 0

  uIsAscending("2004-03-03", "2004-03-05", "2004-03-07")
  // returns 1

  uIsAscending("2004-03-03", "2004-03-07", "2004-03-05")

  //returns 0
  ```

uIsBoolean

Returns **1** if the parameter is **1**, **true**, **yes**or 0, no, false.

Syntax

```
number uIsBoolean(param)
```

Parameters

- **param –** An expressions or value of any data type.

Examples

- **Example 1 –** Check for Boolean value:

  ```
  uIsBoolean("1")     // returns 1
  ```

  ```
  uIsBoolean("yes")   // returns 1
  ```

  ```
  uIsBoolean("true")  // returns 1
  ```

  ```
  uIsBoolean("-1")   // returns 0
  ```

  ```
  uIsBoolean("0")  // returns 1
  ```

uIsDate

Returns **1** if the parameter can be interpreted as a date.

If the second parameter is omitted, the function tries to apply one these formats:

- `y-M-D H:N:S.s`
- `y-M-D H:N:S`
- `y-M-D`
- `H:N:S`

Note: For details about the format string, see the uConvertDate function.

Syntax

```
number uIsDate(datestring [, format])
```

Parameters

- **string datestring –** The string to be checked
- **string format (optional) –** The format of the input date

Examples

- **Example 1 –**

```
uIsDate("2004-02-29") // returns 1
```

```
uIsDate("2003-02-29") // returns 0, since 2003 was not a leap year
```

See also

- *uConvertDate* on page 244

uIsDescending

Returns **1** if every parameter is equal to or lower than its predecessor

Syntax

```
number uIsDescending(params, ...)
```

Parameters

- **params –** A list of expressions or values of any data type

Examples

- **Example 1 –** Check multiple values for a descending order:

```
uIsDescending("C", "B", "A") // returns 1
```

```
uIsDescending("C", "C", "A") // returns 1
```

```
uIsDescending("A", "C", "B") // returns 0
```

```
uIsDescending("3", "2", "1") // returns 1
```

```
uIsDescending("3", "2", "3") // returns 0
```

```
uIsDescending("2004-03-20", "2004-03-15", "2004-03-
07") // returns 1
```

```
uIsDescending("2004-03-20", "2004-03-07", "2004-03-
15") // returns 0
```

uIsEmpty

Returns **1** if the parameter is empty or null

Syntax

```
number uIsEmpty(param)
```

Parameters

- **param –** An expression or value to investigate

Examples

- **Example 1 –**

```
uIsEmpty("1")           // returns 0
```

```
uIsEmpty(null)          // returns 1
```

```
uIsEmpty("")            // returns 1
```

usIsInteger

Returns **1** if the parameter can be interpreted as an integer value

Syntax

```
number uIsInteger(param)
```

Parameters

- **param –** An expression or value to investigate

Examples

- **Example 1 –**

```
uIsInteger ("1")    // returns 1
```

```
uIsInteger ("2.34") // returns 0
```

```
uIsInteger ("ABC")  // returns 0
```

uIsFloat

Returns **1** if the parameter can be interpreted as a floating point value

Syntax

```
number uIsFloat(param)
```

Parameters

- **param –** An expression or value to investigate

Examples

- **Example 1 –**

```
uIsFloat("1")    // returns 1
```

```
uIsFloat("2.34") // returns 1
```

```
uIsFloat("ABC")  // returns 0
```

uIsNull

Returns **1** if the parameter is null

Syntax

```
number uIsNull(param)
```

Parameters

- **param –** An expression or value to investigate

Examples

- **Example 1 –**

  ```
  uIsNull("1")  // returns 0
  ```

  ```
  uIsNull(null) // returns 1
  ```

uIsNumber

Returns **1** if the parameter can be interpreted as a number

Syntax

```
number uIsNumber(param)
```

Parameters

- **param –** An expression or value to investigate

Examples

- **Example 1 –** Check for a numeric value:

  ```
  uIsNumber("1")    // returns 1
  ```

  ```
  uIsNumber("2.34") // returns 1
  ```

  ```
  uIsNumber("ABC")  // returns 0
  ```

Conversion

Review the list of Conversion functions.

uBase64Decode

Decodes a string from a Base64 representation

Syntax

```
string uBase64Decode(input)
```

Parameters

- **string input –** The string to decode

Examples

- **Example 1 –**

```
uBase64Decode("QSBzZWNyZXQ=") // returns "A secret"
```

uBase64Encode

Encodes a string into a Base64 representation

Syntax

```
string uBase64Encode(input)
```

Parameters

- **string input –** The string to encode.

Examples

- **Example 1 –**

```
uBase64Encode("A secret") // returns "QSBzZWNyZXQ="
```

uConvertDate

Converts a date string into the default or a custom date format.

The function handles dates from 1582 to present day. If the date cannot be converted, the result string is empty.

Syntax

```
string uConvertDate(datestring, inputformat [, outputformat])
```

Parameters

- **string datestring –** The date string to convert
- **string inputformat –** The date/time format of the input string
- **string outputformat (optional) –** The desired output format. If omitted, the default format is `y-M-D H:N:S`.

Examples

- **Example 1 –** Convert date strings into a different formats:

```
uConvertDate("2005-06-27 00:00:00","y-M-D H:N:S","D mY") //
returns "27 JUN 05"
```

```
uConvertDate("27 JUN 05", "D m Y") // returns "2005-06-27
00:00:00"
```

Sybase recommends you to provide input data for all the fields required in the output data. Having fields in the output data for which no input data has been provided, can lead to unexpected results. For example, if you require hours, minutes, and seconds fields in the output data, your input data should be something similar to this:

```
uConvertDate("27 JUN 05", "D m Y", "y-m-D 00:00:00") // returns
"2005-06-27 00:00:00"
```

Usage

The function **uConvertDate** converts a date string into a different format using a source and a destination format string. The first parameter is the date string to be converted. The second parameter is a format string, specifying the date format of the input date (see list below). The outputformat parameter is optional; If omitted the date is converted using the format `y-M-D H:N:S`. The function handles dates from 1582 to present day. If the date cannot be converted, the result string is empty.

Identifier	Description
Y	Year 2-digits (06)
y	Year 4-digits (2006)
C	Century (20)
M	Month (03)
m	Month (JUN)
D	Day (12)
H	Hour (00 – 23)
h	Hour (01 – 12)
N	Minutes
n	Month (June)
S	Seconds
s	Hundredth of seconds
t	Thousandth of seconds
A	AM/PM
d	Day of month (05)
E	Day of year (001 – 366)

Identifier	Description
G	Week of year (01 – 52)
F	Week of month (1 – 6)

uFromHex

Converts a hexadecimal value into an integer value

Syntax

```
integer uFromHex(input)
```

Parameters

- **string input –** The string to convert:

Examples

- **Example 1 –**

  ```
  uFromHex("A3F") // returns 2623
  ```

  ```
  uFromHex("B")   // returns 11
  ```

uToHex

Converts an integer value into a hexadecimal value

Syntax

```
string uToHex(input)
```

Parameters

- **integer input –** The integer value to convert:

Examples

- **Example 1 –**

  ```
  uToHex(45) // returns "2D"
  ```

uHexDecode

Composes a string from hexadecimal values

Syntax

```
string uHexDecode(input)
```

Parameters

- **string input –** The hexadecimal string containing the hexadecimal values

Examples

- **Example 1 –** Convert a hexadecimal value to a string:

  ```
  uHexDecode("313730") // returns "170"
  ```

  ```
  uHexDecode(313730)   // returns "170"
  ```

uHexEncode

Encodes the characters of a string into hexadecimal notation

Syntax

```
string uHexEncode(input)
```

Parameters

- **string input –** The string to encode

Examples

- **Example 1 –** Convert a string to hexadecimal values:

  ```
  uHexEncode("170") // returns "313730"
  ```

  ```
  uHexEncode(170)   // returns "313730"
  ```

uToUnicode

Converts a string into its Unicode representation

Syntax

```
string uToUnicode(input)
```

Parameters

- **string input –** The input string

uURIDecode

Decodes a string, replacing escape sequences with their original values

Syntax

```
string uURIDecode(uri)
```

Parameters

- **string uri –** The URI to decode

Examples

- **Example 1 –**

  ```
  uURIDecode(“..www.myServer.com/filename%20with%20spaces.txt") //
  returns
  “..www.myServer.com/filename with spaces.txt"
  ```

uURIEncode

Replaces certain characters in a URI with escape sequences

Syntax

```
string uURIEncode(uri)
```

Parameters

- **string uri –** The URI to encode

Examples

- **Example 1 –**

  ```
  uURIEncode(“..www.myServer.com/filename with spaces.txt")//
  returns
  “..www.myServer.com/filename%20with%20spaces.txt"
  ```

Date and Time

Review the list of Date and Time functions.

Most **Date** and **Time** functions are derived from the **uFormatDate** function. The only difference is that the other **Date** and **Time** functions return only a special format or part of the date and they do not have the first format parameter. Therefore, **uDate()** is equivalent to **uFormatDate("%Y-%m-%d")**.

Time Strings

Learn about the time string function.

Syntax

A time string can be in any of these formats:

1. *YYYY-MM-DD*
2. *YYYY-MM-DD HH:MM*
3. *YYYY-MM-DD HH:MM:SS*

4. *YYYY-MM-DD HH:MM:SS.SSS*
5. *HH:MM*
6. *HH:MM:SS*
7. *HH:MM:SS.SSS*
8. now
9. *DDDD.DDDD*

Note: Formats 5 – 7 that specify only a time assume a date of 2000-01-01. Format 8 is converted into the current date and time, using Universal Coordinated Time (UTC). Format 9 is the Julian day number expressed as a floating point value.

Examples

- **Getting the current time –** If no date is given, the time string **now** is assumed and the date is set to the current date and time.

  ```
  uDate() // returns something like "2006-03-01"
  ```

  ```
  uDate() is equivalent to uDate("now")
  ```
- **Getting a special date –**

  ```
  uDate("2004-01-04 14:26:33")
  // returns the date part "2004-01-04"
  ```

Modifiers

Review the list of modifiers.

Syntax

The time string can be followed by zero or modifiers that alter the date or alter the interpretation of the date. The available modifiers are:

1. *NNN* days
2. *NNN* hours
3. *NNN* minutes
4. *NNN.NNNN* seconds
5. *NNN* months
6. *NNN* years
7. start of month
8. start of year
9. start of day
10. weekday *N*
11. unixepoch
12. localtime
13. utc

Examples

- **Example 1 –** Modifiers 1 – 6 simply add the specified amount of time to the date specified by the preceding time string.

  The "**start of**" modifiers (7 – 9) shift the date backward to the beginning of the current month, year, or day.

  The "**weekday**" (10) modifier advances the date forward to the next date where the weekday number is *N*: Sunday is 0, Monday is 1, and so on.

  The **unixepoch** modifier (11) works only if it immediately follows a time string in the *DDDD.DDDDD* format. This modifier causes the *DDDD.DDDDD* to be interpreted not as a Julian day number as it normally would be, but as the number of seconds since 1970. This modifier allows UNIX-based times to be easily converted to Julian day numbers.

  The **localtime** modifier (12) adjusts the previous time string so that it displays the correct local time. **utc** undoes this.

Date and Time Calculations

These examples show you some typical date and time calculations.

Examples

- **Example 1 –** Compute the current date:

  ```
  uDate('now')
  ```

  Compute the last day of the current month:

  ```
  uDate('now','start of month','+1 month','-1 day')
  ```

  Compute the date and time given a UNIX timestamp 1092941466:

  ```
  uDatetime(1092941466, 'unixepoch')
  ```

  Compute the date and time given a UNIX timestamp 1092941466, and compensate for your local time zone:

  ```
  uDatetime(1092941466, 'unixepoch', 'localtime')
  ```

  Compute the current UNIX timestamp:

  ```
  uFormatDate ('%s','now')
  ```

  Compute the number of seconds between two dates:

  ```
  uJuliandate('now')*86400 - uJuliandate ('2004-01-01
  02:34:56')*86400
  ```

  Compute the date of the first Tuesday in October (January + 9) for the current year:

  ```
  uDate('now','start of year','+9 months','weekday 2')
  ```

Known Limitations

The computation of local time varies by locale. The standard C library function **localtime()** is used to assist in the calculation of local time. Also, the **localtime()** C function normally only works for years between 1970 and 2037.

For dates outside this range, we attempt to map the year into an equivalent year within this range, do the calculation, then map the year back.

- Date computations do not give correct results for dates before Julian day number 0 (-4713-11-24 12:00:00).
- All internal computations assume the Gregorian calendar system.

Date and Time Function List

Review the list of date and time functions.

Function	Description
uDate	Returns a year, month, and day from a date in the format *YYYY-MM-DD*
uDateTime	Returns a year, month, and day from a date in the format *YYYY-MM-DD HH.MM.SS*
uDay	Returns the day number of the date specified
uDayOfYear	Returns the number of days since the beginning of the year
uHour	Returns the hour of the date specified
uQuarter	Returns the quarter of the year
uIsoWeek	Returns the week number defined by ISO 8601
uJuliandate	Returns the number of days since noon in Greenwich on November 24, 4714 B.C, in the format *DDDD.DDDD*
uMinute	Returns the minute of the date specified
uMonth	Returns the month of the name specified
uMonthName	Returns the name of month of the date specified, in the current locale language
uMonthNameShort	Returns the short form of the name of month of the date specified, in the current locale language
uSeconds	Returns the second of the date specified
uTime	Returns the time part from a date in, the format *HH.MM.SS*

Function	Description
uTimeDiffMs	Returns the difference between two dates, in milliseconds
uWeek	Returns the week of the date specified
uWeekday	Returns the weekday of the date specified
uWeekdayName	Returns the weekday name of the date specified, in the current locale language
uWeekdayNameShort	Returns the short form of the weekday name of the date specified in the current locale language
uYear	Returns the year of the date specified

Note: Refer to Date and Time for detailed information about the possible modifier arguments.

uDate

Returns a year, month, and day from a date in the format *YYYY-MM-DD*

Syntax

```
string uDate([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the date part out of a timestamp:

  ```
  uDate("now") // returns current date in the form "YYYY-MM-DD".
  ```

  ```
  uDate("now", "start of year", "9 months", "weekday 2")
  // returns the date of the first Tuesday in October this
  year.
  ```

uDateTime

Returns a year, month, and day from a date in the format *YYYY-MM-DD HH.MM.SS*

Syntax

```
string uDateTime([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the datetime part from a timestamp:

  ```
  uDateTime("now") // returns current date in the form
  "YYYY-MM-DD HH:MM:SS"
  ```

  ```
  uDateTime("now", "start of month", "1 months", "-1 day")
  // returns the date of the last day in this month
  ```

uDay

Returns the day number of the date specified

Syntax

```
string uDay([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the day number out of a timestamp:

  ```
  uDay("now") // returns current day number
  ```

  ```
  uDay("1969-03-13 10:22:23.231") // returns "13"
  ```

uDayOfYear

Returns the number of days since the beginning of the year

Syntax

```
string uDayOfYear([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the day number out of a timestamp:

```
uDayOfYear("now") // returns how many days have already
passed this year
```

```
uDayOfYear("1969-03-13 10:22:23.231") // returns "72"
```

uHour

Returns the hour of the date specified

Syntax

```
string uHour([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uHour("now") // returns current hour
  ```

  ```
  uHour("1969-03-13 10:22:23.231") // returns "10"
  ```

uQuarter

Returns the quarter of the year

Syntax

```
string uQuarter([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uQuarter ("now") // returns current quarter
  ```

  ```
  uQuarter ("2005-03-13 10:22:23.231") // returns "1"
  ```

uIsoWeek

Returns the week number defined by *ISO 8601*

The first week of a year is number 01, which is defined as being the week that contains the first Thursday of the calendar year, which implies that it is also:

- The first week that is mostly within the calendar year
- The week containing January 4th
- The week starting with the Monday nearest to January 1st

The last week of a year, number 52 or 53, therefore is:

- The week that contains the last Thursday of the calendar year
- The last week that is mostly within the calendar year
- The week containing December 28th
- The week ending with the Sunday nearest to December 31st

Syntax

```
number uIsoWeek([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**
  ```
  uIsoWeek("now") // returns current week number
  ```

uJuliandate

Returns the number of days since noon in Greenwich on November 24, 4714 B.C, in the format *DDDD.DDDD*. For date and time calculation, the **juliandate** function is the best choice.

Syntax

```
string uJuliandate([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Convert a date into a numerical value for calculation:
  ```
  uJuliandate("now") // returns current date in the form "DDDD.DDDD"
  ```
  Compute the number of seconds between two dates:
  ```
  uJuliandate('now')*86400 - uJuliandate('2004-01-01
  02:34:56')*86400
  ```

Compute the date and time given a UNIX timestamp 1092941466, and compensate for your local time zone:

```
uJuliandate(1092941466, 'unixepoch', 'localtime')
```

uMinute

Returns the minute of the date specified

Syntax

```
string uMinute([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uMinute("now") // returns current minute
  ```

  ```
  uMinute("1969-03-13 10:22:23.231") //returns "22"
  ```

uMonth

Returns the month of the date specified

Syntax

```
string uMonth([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uMonth("now") // returns current month
  ```

  ```
  uMonth("1969-03-13 10:22:23.231") // returns "03"
  ```

uMonthName

Returns the name of month of the date specified, in the current locale language

Syntax

```
string uMonthName([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the name of month from a date:

  ```
  uMonthName("now") // returns current name of month
  ```

  Set the locale to English:

  ```
  uSetLocale("English")
  uMonthName("1969-03-13 10:22:23.231") // returns "March"
  ```

  Set the locale to German:

  ```
  uSetLocale("German")
  uMonthName("1969-03-13 10:22:23.231") // returns "März"
  ```

uMonthNameShort

Returns the short form of the name of month of the date specified, in the current locale language

Syntax

```
string uMonthNameShort([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the name of month from a date:

  ```
  uMonthNameShort("now") // returns current name of month.
  ```

  Set the locale to English:

  ```
  uSetLocale("English")
  uMonthNameShort("1969-03-13 10:22:23.231") // returns "Mar"
  ```

  Set the locale to German:

  ```
  uSetLocale("German")
  uMonthNameShort("1969-03-13 10:22:23.231") // returns "Mär"
  ```

uSeconds

Returns the second of the date specified.

Syntax

```
string uSeconds([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uSeconds("now") // returns current second
  ```

  ```
  uSeconds("1969-03-13 10:22:23.231") // returns "23"
  ```

uTime

Returns the time part from a date, in the format *HH.MM.SS*.

Syntax

```
string uTime([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Get the time part from a timestamp:

  ```
  uTime() // returns current UTC time
  ```

  ```
  uTime("now","localtime") // returns current local time
  ```

uTimeDiffMs

Returns the difference between two dates, in milliseconds

Syntax

```
string uTimeDiffMs(date1, date2)
```

Parameters

- **string date1 –** The older date
- **string date2 –** The more recent date

Examples

- **Example 1 –**

  ```
  uTimeDiffMs ("18:34:20", "18:34:21")     // returns 1000
  ```

  ```
  uTimeDiffMs ("18:34:20", "18:34:21.200") // returns 1200
  ```

uWeek

Returns the week of the date specified

Syntax

```
string uWeek([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uWeek("now") // returns current week
  ```

  ```
  uWeek("1969-03-13 10:22:23.231") // returns "10"
  ```

uWeekday

Returns the weekday number of the date specified

Syntax

```
string uWeekday([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uWeekday("now") // returns current weekday number
  ```

```
uWeekday("1969-03-13 10:22:23.231") // returns "4" for
Thursday
```

uWeekdayName

Returns the weekday name of the date specified, in the current locale language

Syntax

```
string uWeekdayName([modifiers]);
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uWeekdayName("now") // returns current weekday name
  ```

  Set the locale to English:

  ```
  uSetLocale("English")
  uWeekdayName("1969-03-13 10:22:23.231") // returns "Thursday"
  ```

  Set the locale to German:

  ```
  uSetLocale("German")
  uWeekdayName("1969-03-13 10:22:23.231") // returns "Donnerstag"
  ```

uWeekdayNameShort

Returns the short form of the weekday name of the date specified, in the current locale language

Syntax

```
string uWeekdayNameShort([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uWeekdayNameShort("now") // returns current weekday name
  ```

  Set the locale to English:

  ```
  uSetLocale("English")
  uWeekdayNameShort("1969-03-13 10:22:23.231") //returns "Thu"
  ```

Set the locale to German:

```
uSetLocale("German")
uWeekdayNameShort("1969-03-13 10:22:23.231") //returns "Don"
```

uYear

Returns the year of the date specified

Syntax

```
string uYear([modifiers])
```

Parameters

- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –**

  ```
  uYear("now") // returns current year
  ```

  ```
  uYear("1969-03-13 10:22:23.231") // returns "1969"
  ```

Error Handling

Review the list of Error Handling functions.

uError

Writes an error text into a log and signal an error

Syntax

```
string uError(errortext)
```

Parameters

- **string errortext –** Text to write to log file

Examples

- **Example 1 –** Signal an error:

  ```
  uError("'PP' is no valid country key.")
  ```

uErrortext

Returns the last error message

Syntax

```
string uErrortext()
```

Examples

- **Example 1 –**

```
uErrortext() // returns last error text
```

uInfo

Writes an informal message into a log

Syntax

```
string uInfo(infotext)
```

Parameters

- **string infotext –** Text to write to log file

Examples

- **Example 1 –** Log an informal message:

```
uInfo("21445 records selected.")
```

uWarning

Writes a warning message into a log

Syntax

```
string uWarning(warningtext)
```

Parameters

- **string warningtext –** Text to write to log file

Examples

- **Example 1 –** Log a warning message:

```
uWarning("The attribute for the customer name is null.")
```

uTrace

Writes a trace message into a log.

You must manually set the trace level to at least 1before invoking the **uTrace()** function. To set the trace level to 1, do one of these:

- Invoke **uTracelevel(1)** before invoking the **uTrace()** function.
- If you are using Sybase IQ InfoPrimer Development, set the trace level to 1 in the `Default.ini` file located in the `etc` directory of the installation folder. Restart Sybase IQ InfoPrimer Development.

- If you are using Sybase IQ InfoPrimer Server, start the server with the "**--tracelevel 1**" option.

Syntax

```
string uTrace(tracetext);
```

Parameters

- **string tracetext –** Text to write to log file

Examples

- **Example 1 –**
  ```
  uTrace("CUSTOMER_NAME = " + CUSTOMER_NAME)
  ```

uTracelevel

Sets the detail level of trace messages in the log. The range of **tracelevel** is from 0 (no trace) to 5 (very verbose).

Syntax

```
uTracelevel(tracelevel)
```

Note: Verbose message tracing may dramatically reduce performance.

Parameters

- **integer tracelevel –** Specifies the verbosity of trace messages. (0 = off, 5 = very verbose)

Examples

- **Example 1 –**
  ```
  uTracelevel(5) // sets the tracelevel to 'very verbose'
  ```

Files

Review the list of Conversion functions.

uFileInfo

Returns information about a file. When **infotype** is set to `EXISTS`, the function returns the entire path to the file, if it exists, or an empty string if it does not. If **infotype** set to `SIZE`, the size of the file is returned. If the file does not exist, an empty string is returned.

Note: Use double backslashes in JavaScript environments, because the backslash is used as escape sequence.

Syntax

```
string uFileInfo(file [, infotype])
```

Parameters

- **string file –** The file to investigate
- **string infotype (optional) –** The kind of information to get. Default is `EXISTS`.

Examples

- **Example 1 –** Get file information:

  ```
  uFileInfo("C:\\windows\\notepad.exe") // returns
  C:\windows\notepad.exe
  ```

  ```
  uFileInfo("C:\\windows\\notepad.exe","SIZE") // returns 68608
  ```

uFileWrite

Writes data to a file. If no URL is given, the data is written to a file `write.log` in the Sybase IQ InfoPrimer log directory.

Syntax

```
string uFileWrite(data [, URL] [, append] [, encoding])
```

Parameters

- **string data –** The data to be written
- **string URL (optional) –** URL for file access and location
- **number append (optional) –** Flag (0/1) indicating if the data should be appended or not
- **string encoding (optional) –** The encoding of the target file

Examples

- **Example 1 –** Write data to a file via Common Internet File System (CIFS):

  ```
  uFileWrite("hello", "//myServer/myShare/data/test.txt")
  ```

Formatting

Review the list of Formatting functions.

uFormatDate

Returns a user-defined string with date information.

Syntax

```
number uFormatDate(format, modifiers, ...)
```

Parameters

- **string format –** A format specification for the return string
- **string modifiers (optional) –** List of strings specifying a date or date calculation. Default is the **now** modifier.

Examples

- **Example 1 –** Create a string from a date:

```
uFormatDate("Today is %A the %d of %B in %Y", "now")
//returns something like "Today is Thursday the 10 of
February in 2005"
```

Usage

Special escape sequences in the user-defined format string are replaced by the referring date part.

Escape Sequence	Returns
%A	Weekday name
%a	Weekday name short
%B	Month name
%b	Month name short
%d	Day of month
%f	Fractional seconds SS.SSS
%H	Hour 00 – 24
%j	Day of year 000 – 366
%J	Julian day number
%m	Month
%M	Minute
%s	Seconds since 1970 – 01 – 01
%S	Seconds 00 – 59
%w	Day of week 0 – 6, 0=Sunday
%W	Week of year
%Y	Year 0000 – 9999
%%	%

Fuzzy Search

Review the list of Fuzzy Search functions.

uGlob

Compares case-sensitive values that are similar, using the UNIX file globbing syntax for its wildcards.

Syntax

```
bool uGlob(pattern, text)
```

Parameters

- **string pattern –** A string describing a match pattern
- **string text –** A string to investigate

Examples

- **Example 1 –** Compare values using UNIX file globbing syntax:

  ```
  uGlob("Mr. *", "Mr. Smith")   // returns 1, indicating
  a match
  ```

  ```
  uGlob("Mr. *", "Mrs. Clarke") // returns 0
  ```

  Globbing rules:

  “*” – matches any sequence of zero or more characters.

  “?” – matches exactly one character. [^...] – matches one character not in the enclosed list.

  [...] – matches one character from the enclosed list of characters.

  With [...] and [^...] matching, a closing square bracket ( ]) can be included in the list by making it the first character after an opening square bracket ([ ) or a caret (^). Specify a range of characters using a hyphen (-):

  - “[a-z]” matches any single lowercase letter. To match a hyphen (-), make it the last character in the list.
  - To match an asterisk (*) or a question mark (?), place them in square brackets ([]).
    For example: abc[*]xyz, matches the literal value “abc*xyz” .

uLike

Compares values case insensitive.

The **uLike** function performs a pattern-matching comparison. The first parameter contains the pattern, the second parameter contains the string to match against the pattern. A percent symbol (%) in the pattern matches any sequence of zero or more characters in the string. An underscore (_) in the pattern matches any single character in the string. Any other character matches itself or its lowercase or uppercase equivalent.

Note: Currently, **uLike** only interpret only uppercase and lowercase for 7-bit Latin characters, which means **uLike** is case-sensitive for 8-bit ISO8859 characters or UTF-8 characters. For example: `uLike('a' ,'A')` returns 1. `uLike('æ' ,'Æ')` returns 0.

Syntax

```
number uLike(pattern, text)
```

Parameters

- **string pattern –** A string describing a match pattern
- **string text –** A string to investigate

Examples

- **Example 1 –** Compare values using pattern matching:

  ```
  uLike("% happy %", "A happy man.") // returns 1
  ```

  ```
  uLike("% happy %", "A sad man.")   // returns 0
  ```

uMatches

Returns true if a given string matches a regular expression.

Syntax

```
number uMatches(text, regexpr)
```

Parameters

- **string text –** Text to investigate
- **string regexpr –** Regular expression specification

Examples

- **Example 1 –** Check if a string could be interpreted as a floating point number:

  ```
  uMatches("abc","[-+]?[0-9]*\\.?[0-9]*")  // returns 0
  ```

  ```
  uMatches("1.23","[-+]?[0-9]*\\.?[0-9]*")  // returns 1
  ```

Lookup

Review the list of Lookup functions.

uChoice

Returns the value of a given parameter specified by an index. The index value is zero-based, so an index of zero returns the second parameter.

Syntax

```
string uChoice(index, values, ...)
```

Parameters

- **integer index –** Zero based index number referencing the return value.
- **string values –** List of values

Examples

- **Example 1 –** IF construct:

  ```
  uChoice(0, "A", "B") // returns "A"
  ```

  ```
  uChoice(1, "A", "B") // returns "B"
  ```

  CASE construct:

  ```
  uChoice(2, "n.a.", "Jan", "Feb", "Mar") //returns "Feb"
  ```

  Simulate a lookup function, where you want to replace a color ID with a corresponding color name:

  ```
   uChoice(IN.Color, "n.a.", "Red", "Blue", "Green")
  ```

uFirstDifferent

Returns the first parameter value that differs from the first parameter

Syntax

```
string uFirstDifferent(params, ...)
```

Parameters

- **params –** A list of expressions or values of any data type

Examples

- **Example 1 –**

  ```
  uFirstDifferent("2004-05-01", "2004-05-01", "2005-01-04",
  "2005-11-24",) //returns "2005-01-04"
  ```

uFirstNotNull

Returns the first non-null parameter.

Syntax

```
string uFirstNotNull(params, ...)
```

Parameters

- **params –** A list of expressions or values of any data type

Examples

- **Example 1 –**

  ```
  uFirstNotNull(null, null , "A", "B") // returns "A"
  ```

uElements

Returns the number of elements in a delimited string. If the second parameter is omitted, a space (ASCII 32) is taken as a delimiter.

Syntax

```
integer uElements(text [, delimiter])
```

Parameters

- **string text –** A string to investigate
- **string delimiter (optional) –** The delimiter to be used. Default delimiter is a space character.

Examples

- **Example 1 –** Count tokens in a delimited string:

  ```
  uElements("James T. Kirk") // returns 3
  ```

uToken

Returns the Nth element from a delimited string. The second parameter specifies the token number. The index starts at 1. If the third parameter is omitted, a space (ASCII 32) is used as the delimiter.

Syntax

```
string uToken(text, index [, delimiter])
```

Parameters

- **string text –** A string to investigate
- **Integer index –** Number of tokens to be returned
- **string delimiter (optional) –** The delimiter to be used. Default delimiter is a space character.

Examples

- **Example 1 –**

```
uToken("James T. Kirk", 1) // returns "James"
```

```
uToken("James T. Kirk", 2) // returns "T."
```

Miscellaneous

Review the list of miscellaneous functions.

uCommandLine

Returns the command line string of the current process

Syntax

```
string uCommandLine()
```

Examples

- **Example 1 –**

```
uCommandLine() // returns
"GridNode.exe --port 5124"
```

Note: uCommandLine is not supported on UNIX.

uGetEnv

Returns the value of an environment variable

Syntax

```
string uGetEnv(variable)
```

Parameters

- **string variable –** Name of the environment variable to read

Examples

- **Example 1 –**

```
uGetEnv("LOAD_MAX_VALUE")
```

uGuid

Returns a global unique identifier in one of the specified formats.

- *numeric* – digits only
- *base64* – Base64-encoded
- *hex* – hex format without hyphens

Syntax

```
string uGuid([format])
```

Parameters

- **string format (optional) –** Format for the GUID value to be returned

Examples

- **Example 1 –**

  ```
  uGuid() // returns for example
  A8A10D9F-963F-4914-8D6FC8527A50EF2A
  ```

uMD5

Generates a checksum over a given string with a fixed length of 32 characters

Syntax

```
string uMD5(text)
```

Parameters

- **string text –** Text to build a checksum on

Examples

- **Example 1 –**

  ```
  uMD5("Austin Powers") // returns
  "C679A893E3DA2CC0741AC7F527B1D4EB"
  ```

uScriptLoad

Loads and evaluates JavaScript and returns the result

Syntax

```
string uScriptLoad(filelocation)
```

Parameters

- **string filelocation –** The JavaScript file to load.

Examples

- **Example 1 –** Load an external JavaScript file:

```
uScriptLoad("\\server3\myScripts\basicFunctions.js")
```

uSetEnv

Set the value of an environment variable

Syntax

```
string uSetEnv(variable, value)
```

Parameters

- **string variable –** Name of the environment variable to set
- **string value –** Value to set

Examples

- **Example 1 –**

```
uSetEnv("LOAD_MAX_VALUE", IN.Date)
```

uSetLocale

Changes the locale date and time settings to a different language

Syntax

```
string uSetLocale([language] [, country] [, codepage])
```

Parameters

- **string language (optional) –** Language string to be used (see table in Usage section)
- **string country (optional) –** Country name to be used (see table in Usage section)
- **string codepage (optional) –** Code page number as string

Examples

- **Example 1 –** Retrieve month names in different languages:

```
locale:uSetLocale("english")    // switch to english
uMonthName("2005-03-22") // returns "March"
uSetLocale("german")     // switch to german
```

```
uMonthName("2005-03-22") // returns "Marz"
uSetLocale("C")          // switch back to OS default
```

Usage for uSetLocale

The language strings in the table below are recognized. Any language not supported by the operating system is not accepted by **uSetLocale**.

Language Strings

Note: The three-letter language-string codes are valid only in Windows NT and Windows 95.

Primary Language	Sublanguage	Language String
Chinese	Chinese	"chinese"
Chinese	Chinese (simplified)	"chinese-simplified" or "chs"
Chinese	Chinese (traditional)	"chinese-traditional" or "cht"
Czech	Czech	"csy" or "czech"
Danish	Danish	"dan"or "danish"
Dutch	Dutch (Belgian)	"belgian", "dutch-belgian",or "nlb"
Dutch	Dutch (default)	"dutch" or "nld"
English	English (Australian)	"australian", "ena", or "english-aus"
English	English (Canadian)	"canadian", "enc", or "english-can"
English	English (default)	"english"
English	English (New Zealand)	"english-nz" or "enz"
English	English (UK)	"eng", "english-uk", or "uk"
English	English (USA)	english", "americanenglish", "english-american", "english-us", "english-usa", "enu", "us", or "usa"
Finnish	Finnish	"fin" or "finnish"
French	French (Belgian)	"frb" or "french-belgian"
French	French (Canadian)	"frc" or "frenchcanadian"

Primary Language	Sublanguage	Language String
French	French (default)	"fra"or "french"
French	French (Swiss)	"french-swiss" or "frs"
German	German (Austrian)	"dea" or "germanaustrian"
German	German (default)	"deu" or "german"
German	German (Swiss)	"des", "german-swiss", or "swiss"
Greek	Greek	"ell" or "greek"
Hungarian	Hungarian	"hun" or "hungarian"
Icelandic	Icelandic	"icelandic" or "isl"
Italian	Italian (default)	"ita" or "italian"
Italian	Italian (Swiss)	"italian-swiss" or "its"
Japanese	Japanese	"japanese" or "jpn"
Korean	Korean	"kor" or "korean"
Norwegian	Norwegian (Bokmal)	"nor" or "norwegianbokmal"
Norwegian	Norwegian (default)	"norwegian"
Norwegian	Norwegian (Nynorsk)	"non" or "norwegiannynorsk"
Polish	Polish	"plk" or "polish"
Portuguese	Portuguese (Brazil)	"portuguese-brazilian" or "ptb"
Portuguese	Portuguese (default)	"portuguese" or "ptg"
Russian	Russian (default)	"rus" or "russian"
Slovak	Slovak	"sky" or "slovak"
Spanish	Spanish (default)	"esp" or "spanish"
Spanish	Spanish (Mexican)	"esm" or "spanish-mexican"
Spanish	Spanish (Modern)	"esn" or "spanish-modern"
Swedish	Swedish	"sve" or "swedish"
Turkish	Turkish	"trk" or "turkish"

Country or Region Strings

The following is a list of country/regions strings recognized by **uSetLocale**. Strings for countries/regions that are not supported by the operating system are not accepted by **uSetLocale**. Three-letter country or region codes are from ISO or IEC (International Organization for Standardization, International Electrotechnical Commission) specification 3166.

Country or Region	Country or Region String
Australia	"aus" or "australia"
Austria	"austria" or "aut"
Belgium	"bel" or "belgium"
Brazil	"bra" or "brazil"
Canada	"can" or "canada"
Czech Republic	"cze" or "czech"
Denmark	"denmark" or "dnk"
Finland	"fin" or "finland"
France	"fra" or "france"
Germany	"deu" or "germany"
Greece	"grc" or "greece"
Hong Kong SAR	"hkg", "hong kong", or "hong-kong"
Hungary	"hun" or "hungary"
Iceland	"iceland" or "isl"
Ireland	"ireland" or "irl"
Italy	"ita" or "italy"
Japan	"japan" or "jpn"
Korea	"kor", "korea"
Mexico	"mex" or "mexico"
Netherlands	"nld", "holland", or "netherlands"
New Zealand	"new zealand", "new-zealand", "nz", or "nzl"
Norway	"nor" or "norway"

Country or Region	Country or Region String
People's Republic of China	"china", "chn", "pr china", or "pr-china"
Poland	"pol" or "poland"
Portugal	"prt" or "portugal"
Russia	"rus" or "russia"
Singapore	"sgp" or "singapore"
Slovak Repubic	"svk" or "slovak"
Spain	"esp" or "spain"
Sweden	"swe" or "sweden"
Switzerland	"che" or "switzerland"
Taiwan	"taiwan" or "twn"
Turkey	"tur" or "turkey"
United Kingdom	"britain", "england", "gbr", "great britain", "uk", "united kingdom", or "united-kingdom"
United States of America	"america", "united states", "unitedstates", "us", or "usa"

uSleep

Suspends the process for a specified number of milliseconds

Syntax

```
string uSleep(msecs)
```

Parameters

- **integer msecs –** Number of milliseconds for which to suspend the process

Examples

- **Example 1 –**
  ```
  uSleep(1000) // suspends the process for one second
  ```

uSystemFolder

Returns predefined application and system paths

Syntax

```
string uSystemFolder([foldertype])
```

Examples

- **Example 1 –**
```
uSystemFolder("APP_LOG")  // returns the path to the log directory
```

Usage

Use **uSystemFolder** to access a special directory on the file system. You can specify the following folders:

Group	Name	Description
Application	APP_MAIN	The base application path. A typical path is `C:\Program Files\ETL`.
Application	APP_LIB	Shared library directory. A typical path is `application_directory\lib`.
Application	APP_LOG	Shared library directory. A typical path is `application_directory\lib`.
Application	APP_CONFIG	Config file directory. A typical path is `application_directory\etc`.
Application	APP_LICENSE	License directory. A typical path is `application_directory\license`.
Application	APP_SCRIPT	Script directory. A typical path is `application_directory\scripts`.
Application	APP_GRAMMAR	Grammar directory. A typical path is `application_directory\grammar`.
Application	APP_LANGUAGE	Language file directory. A typical path is `application_directory\language`.

Group	Name	Description
Application	APP_DATABASE	Database directory. A typical path is `application_directory\database`.
Application	APP_TEMP	Temporary directory. A typical path is `application_directory\temp`.
Application	APP_DEMODATA	Demodata directory. A typical path is `application_directory\demodata`.
Application	APP_USERDATA	Directory where user specific files are stored. Typical path is `C:\Documents and Settings\username\Application Data\ETL`.
Windows	ALTSTARTUP	The file system directory that corresponds to the user's non-localized Startup program group.
Windows	APPDATA	The file system directory that serves as a common repository for application-specific data. A typical path is `C:\Documents and Settings\username\Application Data`.
Windows	CDBURN_AREA	The file system directory acting as a staging area for files waiting to be written to CD. A typical path is `C:\Documents and Settings\username\Local Settings\Application Data\Microsoft\CD Burning`.
Windows	COMMON_ADMIN-TOOLS	The file system directory containing administrative tools for all users of the computer
Windows	COMMON_APPDATA	The file system directory containing application data for all users. A typical path is `C:\Documents and Settings\All Users\Application Data`.

Group	Name	Description
Windows	COMMON_DESKTOPDIR-ECTORY	The file system directory that contains files and folders that appear on the desktop for all users. A typical path is `C:\Documents and Settings\All Users\Desktop`. Valid only for Windows NT systems.
Windows	COMMON_DOCUMENTS	The file system directory that contains documents that are common to all users. A typical paths is `C:\Documents and Settings\All Users\Documents`.
Windows	COMMON_FAVORITES	The file system directory that serves as a common repository for favorite items common to all users. Valid only for Windows NT systems.
Windows	COMMON_MUSIC	The file system directory that serves as a repository for music files common to all users. A typical path is `C:\Documents and Settings\All Users \Documents\My Music`.
Windows	COMMON_PICTURES	The file system directory that serves as a repository for image files common to all users. A typical path is `C:\Documents and Settings\All Users \Documents\My Pictures`.
Windows	COMMON_PROGRAMS	The file system directory that contains the directories for the common program groups that appear on the Start menu for all users. A typical path is `C:\Documents and Settings\All Users \Start Menu\Programs`. Valid only for Windows NT systems.
Windows	COMMON_STARTMENU	The file system directory that contains the programs and folders that appear on the Start menu for all users. A typical path is `C:\Documents and Settings \All Users\Start Menu`. Valid only for Windows NT systems.

Group	Name	Description
Windows	COMMON_STARTUP	The file system directory that contains the programs that appear in the Startup folder for all users. A typical path is `C:\Documents and Settings\All Users\Start Menu\Programs\Startup`. Valid only for Windows NT systems.
Windows	COMMON_TEMPLATES	The file system directory that contains the templates that are available to all users. A typical path is `C:\Documents and Settings\All Users\Templates`. Valid only for Windows NT systems.
Windows	COMMON_VIDEO	The file system directory that serves as a repository for video files common to all users. A typical path is `C:\Documents and Settings\All Users\Documents\My Videos`.
Windows	COOKIES	The file system directory that serves as a common repository for Internet cookies. A typical path is `C:\Documents and Settings\username\Cookies`.
Windows	DESKTOP	The virtual folder representing the Windows desktop, the root of the namespace.
Windows	DESKTOPDIRECTORY	The file system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself). A typical path is `C:\Documents and Settings\username\Desktop`.
Windows	FAVORITES	The file system directory that serves as a common repository for the user's favorite items. A typical path is `C:\Documents and Settings\username\Favorites`.
Windows	FONTS	A virtual folder containing fonts. A typical path is `C:\Windows\Fonts`.
Windows	HISTORY	The file system directory that serves as a common repository for Internet history items.

Group	Name	Description
Windows	INTERNET_CACHE	The file system directory that serves as a common repository for temporary Internet files. A typical path is `C:\Documents and Settings\username\Local Settings\Temporary Internet Files`.
Windows	MYDOCUMENTS	Virtual folder representing the My Documents desktop item.
Windows	MYMUSIC	The file system directory that serves as a common repository for music files. A typical path is `C:\Documents and Settings\User\My Documents\My Music`.
Windows	MYPICTURES	The file system directory that serves as a common repository for image files. A typical path is `C:\Documents and Settings\username\My Documents\My Pictures`.
Windows	MYVIDEO	The file system directory that serves as a common repository for video files. A typical path is `C:\Documents and Settings\username\My Documents\My Videos`.
Windows	NETHOOD	A file system directory containing the link objects that may exist in the My Network Places virtual folder. It is not the same as `CSIDL_NETWORK`, which represents the network namespace root. A typical path is `C:\Documents and Settings\username\NetHood`.
Windows	PERSONAL	The virtual folder representing the My Documents desktop item. This is equivalent to MYDOCUMENTS.
Windows	PRINTHOOD	The file system directory that contains the link objects that can exist in the Printers virtual folder. A typical path is `C:\Documents and Settings\username\PrintHood`.

Group	Name	Description
Windows	PROFILE	The user's profile folder. A typical path is `C:\Documents and Settings\username`. Applications should not create files or folders at this level; they should instead place data under the locations referred to by `APPDATA` or `LOCAL_APPDATA`.
Windows	PROGRAM_FILES	The Program Files folder. A typical path is `C:\Program Files`.
Windows	PROGRAM_FILES_COMMON	A folder for components that are shared across applications. A typical path is `C:\Program Files\Common`. Valid only for Windows NT, Windows 2000, and Windows XP systems.
Windows	PROGRAMS	The file system directory that contains the user's program groups (which are themselves file system directories). A typical path is `C:\Documents and Settings\username\Start Menu\Programs`.
Windows	RECENT	The file system directory that contains shortcuts to the user's most recently used documents. A typical path is `C:\Documents and Settings\username\My Recent Documents`. To create a shortcut in this folder, use `SHAddToRecentDocs`. In addition to creating the shortcut, this function updates the shell's list of recent documents and adds the shortcut to the My Recent Documents submenu of the Start menu.
Windows	SENDTO	The file system directory that contains Send To menu items. A typical path is `C:\Documents and Settings\username\SendTo`.
Windows	STARTMENU	The file system directory containing Start menu items. A typical path is `C:\Documents and Settings\username\Start Menu`.

Group	Name	Description
Windows	STARTUP	The file system directory that corresponds to the user's Startup program group. The system starts these programs whenever any user logs in to Windows NT or starts Windows 95. A typical path is `C:\Documents and Settings\username\Start Menu\Programs\Startup`.
Windows	SYSTEM	The Windows System folder. A typical path is `C:\Windows\System32`.
Windows	TEMPLATES	The file system directory that serves as a common repository for document templates. A typical path is `C:\Documents and Settings\username\Templates`.
Windows	WINDOWS	The Windows directory or SYSROOT. This corresponds to the `%windir%` or `%SYSTEMROOT%` environment variables. A typical path is `C:\Windows`.

Network

Review the list of Network functions.

uHostname

Returns the local network name

Syntax

```
string uHostname()
```

Examples

- **Example 1 –**

```
uHostname()  // returns something like “pollux” or “castor”
```

uSMTP

Sends a mail to a SMTP server

Syntax

```
bool uSMTP(serverURL, sender, recipients, subject, body)
bool uSMTP(sender, recipients, subject, body)
bool uSMTP(recipients, subject, body)bool uSMTP(subject, body)
bool uSMTP(body)
```

Parameters

- **string serverURL –** URL for specifying the SMTP server, port, user name, and password to use
- **string sender –** E-mail address of sender
- **string recipients –** Comma-separated list of recipients
- **string subject –** Subject of the e-mail message
- **string body –** Content of the e-mail message

Examples

- **Example 1 –**

  ```
  uSMTP("Just a mail")uSMTP("Testmail!", "Just a mail")
  ```

Usage

The **uSMTP** function allows e-mail messages to be sent to multiple recipients using a SMTP server.

The server URL for specifying the SMTP server uses this syntax: `protocol://user:password@server:port`

Protocol can be one of:

<empty> – SMTP with SSL encryption, if applicable

SMTP – SMTP without SSL encryption

SMTPS – SMTP with SSL encryption

User name and password – The user name and password are used to authenticate the client. If not provided, no authentication is performed.

Note: If the user name contains a @ sign, replace it with # to avoid ambiguities.

Port – TCP port to be used, default is 25.

```
myServer
myServer:123
SMTPS://myServer:123
Me:secret@myServer
```

Specify recipients by adding a list of addresses separated by a comma. By default, all recipients are addressed directly. To send a carbon copy or blind carbon copy, simply add "cc:" or "bcc:" before the e-mail address:

```
user@host.domain
My Name <user@host.domain>
To: My Name <user@host.domain>
```

```
To: user@host.domain
Cc: My Name <user@host.domain>
To: user@host.domain, Bcc: Test User <test@myserver.com>
```

If the SMTP server allows encrypted communication, it is performed automatically. If you provide user name and password, authentication methods are tried in the following sequence: **PLAIN**, **LOGIN**.

You can specify your personal defaults in the `INI` file:

```
[SMTP]
ServerURL=<your default server URL>
Sender=<your default sender>
Recipients=<your default recipients>
Subject=<your default subject>
```

For example:

```
[SMTP]
ServerURL=maxm:secret@mail.gmail.com
Sender= Maxi <Max.Mustermann@ gmail.com>
Recipients=ETLAdmin@MyCompany.com, Cc: QA qa@MyCompany.com
Subject=ETL Message
```

Numeric

Review the list of Numeric functions.

uAbs

Returns the magnitude of a real number, ignoring its positive or negative sign

Syntax

```
number uAbs(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**

  ```
  uAbs(1522)      // returns 1522
  ```

  ```
  uAbs('-123.45') // returns 123.45
  ```

  ```
  uAbs('123ABC')  // returns 0
  ```

uCeil

Returns least integer greater than or equal to argument

Syntax

```
number uCeil(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –** Round up numbers:

  ```
  uCeil(1523.1) // returns 1524
  ```

  ```
  uCeil(1523.9) // returns 1524
  ```

uDiv

Returns the division integer

Syntax

```
number uDiv(value, divisor)
```

Parameters

- **number value –** A number to calculate on
- **number divisor –** The divisor of the division

Examples

- **Example 1 –**

  ```
  uDiv(10, 3) // returns 3
  ```

uExp

Returns the exponential, base e

Syntax

```
number uExp(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**
  ```
  uExp(1) // returns "2.718281828459045"
  ```

uFloor
Returns greatest integer less than or equal to argument

Syntax

```
number uFloor(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**
  ```
  uFloor(1523.1) // returns 1523
  ```
  ```
  uFloor(1523.9) // returns 1523
  ```

uLn
Returns the natural logarithm (base e) of a number

Syntax

```
number uLn(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**
  ```
  uLn(2.718281828) // returns 0.999999
  ```

uLog
Returns the logarithm of a number

Syntax

```
number uLog(value [, base])
```

Parameters

- **number value –** A number to calculate on
- **number base (optional) –** The base for the logarithm. If omitted, a base of 10 is used.

Examples

- **Example 1 –**

  ```
  uLog(100) // returns 2
  ```

  ```
  uLog(16, 2) // returns 4
  ```

uMod

Returns the modulo of division

Syntax

```
number uMod(value, divisor)
```

Parameters

- **number value –** A number to calculate on
- **number divisor –** The divisor of the division

Examples

- **Example 1 –**

  ```
  uMod(10, 3) // returns 1
  ```

uPow, uPower

Returns the value of a base expression taken to a specified power

Syntax

```
number uPow(value, exponent)
```

Parameters

- **number value –** A number to calculate on
- **number exponent –** A number to be used as the exponent

Examples

- **Example 1 –**

  ```
  uPow(10, 3) // returns 1000
  ```

uRandom
Returns a random number

Syntax
```
number uRandom()
```

Examples

- **Example 1 –** Random numbers
  ```
  uRandom() // returns a value like "0.696654639123727"
  ```

uRound
Returns the rounded argument to nearest integer

Syntax
```
number uRound(value [, scale])
```

Parameters

- **number value –** A number to calculate on
- **number scale (optional) –** Number of digits

Examples

- **Example 1 –**
  ```
  uRound(10.1) // returns "10"
  ```
  ```
  uRound(10.49) // returns "10"
  ```
  ```
  uRound(10.5) // returns "11"
  ```
  ```
  uRound(10.9) // returns "11"
  ```
  ```
  uRound(1.235, 2) // returns "1.24"
  ```

uSgn
Returns the sign of a given value

Syntax
```
number uSgn(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**

```
uSgn(-10.4) // returns -1
uSgn(0)     // returns 0
uSgn(10.4)  // returns 1
uSgn(null)  // returns null
```

uSqrt

Returns the square root of a given value

Syntax

```
number uSqrt(value)
```

Parameters

- **number value –** A number to calculate on

Examples

- **Example 1 –**

```
uSqrt(25)   // returns 5
uSqrt(0)    // returns 0
uSqrt(null) // returns null
```

Script

Review the list of Script functions.

uEvaluate

Evaluates a function or JavaScript expression and returns the result

Syntax

```
string uEvaluate(expression)
```

Parameters

- **Number expression –** JavaScript code to evaluate

Examples

- **Example 1 –** Evaluate functional expressions:

```
uEvaluate("3 + 5")
uEvaluate("parseFloat(IN.Salary) + 1500")
```

Define custom functions:

```
uEvaluate("function timesTwo(a){ return a*2; }")
```

Use custom functions:

```
uEvaluate("timesTwo(4)")
```

```
uEvaluate("timesTwo(IN.Salary)")
```

Evaluate scripts:

```
uEvaluate("if ("parseFloat(IN.Salary) > 2000") {2000;}
else {("parseFloat(IN.Salary) + 500");}")
```

String

Review the list of String functions.

uAsc, uUnicode

Returns unicode character value of a specified character.

Syntax

```
number uAsc(value [, index])
```

Parameters

- **string value –** An input string
- **number index (optional) –** Character position for reading ASCII value

Examples

- **Example 1 –**

```
uAsc("Big Ben")    // returns 66
```

```
uAsc("Big Ben", 2) // returns 105
```

uChr, uUniChr

Returns the Unicode string corresponding to the given number, or formats a string

Syntax

```
string uChr(params, ...)
```

Parameters

- **params –** A list of expressions or values

Examples

- **Example 1 –**

```
uChr(64) // returns "@"
```

```
uChr("\u0064\u006f\u0067") // returns "dog"
```

```
uChr(65,"pple") // returns "apple"
```

uCap

Returns the capitalized representation of a string. In other words, the first letter of each word in the string is capitalized.

Syntax

```
string uCap(text)
```

Parameters

- **Input text –** The string to be capitalized

Examples

- **Example 1 –**

```
uCap('fArmeR, ASTROnaut') // returns 'Farmer, Astronaut'
```

```
uCap('the first weekend') // returns 'The First Weekend'
```

uCon, uConcat

Concatenates all given parameters into a single string

Syntax

```
string uConcat(params)
```

Parameters

- **params –** A list of expressions or values of any data type

Examples

- **Example 1 –**

```
uConcat("For ", 3, " years.") returns "For 3 years."
```

uJoin

Concatenates a delimited string with special null and empty value handling

Syntax

```
string uJoin(delimiter, allowEmpty, params, ...)
```

Parameters

- **string delimiter –** Delimiter to be used between all other string parts
- **number allowEmpty –** Flag (0/1) that indicate whether empty fields are allowed
- **string params –** List of strings to concatenate

Examples

- **Example 1 –**

  ```
  uJoin("-", 1, "James", "", "Tiberius", "Kirk") //
  returns "James--Tiberius-Kirk"
  ```

  ```
  uJoin("-", 0, "James", "", "Tiberius", "Kirk") //
  returns "James-Tiberius-Kirk"
  ```

uLeft

Returns the leftmost **N** characters from a string

Syntax

```
string uLeft(input, chars)
```

Parameters

- **string input –** The input string
- **number chars –** The number of characters to be retrieved

Examples

- **Example 1 –**

  ```
  uLeft("James T. Kirk", 5) // returns "James"
  ```

  ```
  uLeft(null, 5)            // returns null
  ```

uLength, uLen

Returns the length of a string

Syntax

```
number uLength(input)
```

Parameters

- **string input –** The input string

Examples

- **Example 1 –**
  ```
  uLength("James T. Kirk") // returns 13
  ```

uSubstr, uMid

Returns a part of a string

Syntax

```
string uSubstr(input, position, length)
```

Parameters

- **string input –** Input string
- **number position –** The position from where to start reading
- **number length –** The number of characters to read

Examples

- **Example 1 –**
  ```
  uSubstr("James T. Kirk", 7, 2) // returns "T."
  ```

uLPos

Find the first position of a substring within a string. A result of zero indicates that the substring has not been found.

Syntax

```
string uLPos(input, substring)
```

Parameters

- **string input –** The input string
- **string substring –** The substring to search

Examples

- **Example 1 –**
  ```
  uLPos("James T. Kirk", "T") //returns 7
  ```

uLower, uLow

Returns the input string in lower case letters

Syntax

```
string uLower(input)
```

Parameters

- **string input –** The string to convert

Examples

- **Example 1 –**

  ```
  uLower("James T. Kirk") // returns "james t. kirk"
  ```

uLStuff

Fills the left side of a string up to specified length

Syntax

```
string uLStuff(input, length [, stuff])
```

Parameters

- **string input –** The string to stuff
- **number length –** New length of string
- **string stuff (optional) –** String to append, default is an empty space (ASCII 32)

Examples

- **Example 1 –**

  ```
  uLStuff("3.5", 5)      // returns "  3.5"
  ```

  ```
  uLStuff("3.5", 5, "0") // returns "003.5"
  ```

uLTrim

Removes characters from the left side of the string. If the second parameter is omitted, it defaults to a space character (ASCII 32).

Syntax

```
string uLTrim(input, trimstring)
```

Parameters

- **string input –** The string to be trimmed
- **string trimstring –** The string to trim

Examples

- **Example 1 –**

```
uLTrim(" 3.5")       // returns "3.5"
```

```
uLTrim("003.5", "0") // returns "3.5"
```

uRepeat

Returns the given string repeated **N** times

Syntax

```
string uRepeat(input, repeats)
```

Parameters

- **string input –** The string to be repeated
- **number repeats –** The number of times to repeat the input string

Examples

- **Example 1 –**

```
uRepeat("Hello ", 4) // returns "Hello Hello Hello Hello "
```

uReplace

Replaces parts of a string

Syntax

```
string uReplace(input, search, replace)
```

Parameters

- **string input –** The string to be worked on
- **string search –** The pattern to be searched
- **string replace –** The string to replace any match

Examples

- **Example 1 –**

```
uReplace("At four o' clock he became four", "four", "4")
// returns "At 4 o' clock he became 4"
```

uReverse

Reverses a string

Syntax

```
string uReverse(input)
```

Parameters

- **string input –** The string to reverse

Examples

- **Example 1 –**

  ```
  uReverse("Smith")  // returns "htimS"
  ```

uRight

Returns the rightmost **N** characters from a string

Syntax

```
string uRight(input, chars)
```

Parameters

- **string input –** The input string
- **number chars –** The number of characters to be read

Examples

- **Example 1 –**

  ```
  uRight("James T. Kirk", 4) // returns "Kirk"
  ```

  ```
  uRight(null, 5) / / returns null
  ```

uRPos

Find the last position of a substring within a string

Syntax

```
string uRPos(input, substring)
```

Parameters

- **string input –** The input string
- **string substring –** The substring to find

Examples

- **Example 1 –** Find the last ocurrence of a substring:

  ```
  uRPos("James T. Kirk", "T") //returns 7
  ```

uRStuff

Fills the right side of a string up to specified length

Syntax

```
string uRStuff(input, length [, stuffstring])
```

Parameters

- **string input –** The input string
- **number length –** The new length of the result string
- **string stuffstring (optional) –** The string to append

Examples

- **Example 1 –**

  ```
  uRStuff("3.5", 5)      // returns "3.5  "
  ```

  ```
  uRStuff("3.5", 5, "0") // returns "3.500"
  ```

uRTrim

Removes characters from the right side of the string

Syntax

```
string uRTrim(input [, trimstring])
```

Parameters

- **string input –** The input string
- **string trimstring (optional) –** The string to trim

Examples

- **Example 1 –**

  ```
  uRTrim("3.5 ")       // returns "3.5"
  ```

  ```
  uRTrim("3.500", "0") // returns "3.5"
  ```

uTrim

Removes characters from both sides of the string

Syntax

```
string uTrim(input [, trimstring])
```

Parameters

- **string input –** The input string
- **string trimstring (optional) –** The string to trim

Examples

- **Example 1 –**

  ```
  uTrim(" 3.5 ")        // returns "3.5"
  ```

  ```
  uTrim("003.500", "0") // returns "3.5"
  ```

uUpper, uUpp

Returns the input string in upper case letters

Syntax

```
string uUpper(input)
```

Parameters

- **string input –** The input string

Examples

- **Example 1 –**

  ```
  uUpper("James T. Kirk") // returns "JAMES T. KIRK"
  ```

Trigonometric

Review the list of Trigonometric functions.

uAcos

Returns the arccosine (in radians) of a number

Syntax

```
number uAcos(value)
```

Parameters

- **number value –** The input value

uAsin

Returns the arcsine (in radians) of a number

Syntax

```
number uAsin(value)
```

Parameters

- **number value –** The input value

uAtan

Returns the arctangent (in radians) of a number

Syntax

```
number uAtan(value)
```

Parameters

- **number value –** The input value

uCos

Returns the cosine (in radians) of a number

Syntax

```
number uCos(value)
```

Parameters

- **number value –** The input value

uSin

Returns the sine (in radians) of a number

Syntax

```
number uSin(value)
```

Parameters

- **number value –** The input value

uTan

Returns the tangent (in radians) of a number

Syntax

```
number uTan(value)
```

Parameters

- **number value –** The input value

Supported Sybase IQ Functions for SQL Transformation Project

List of Sybase IQ functions that can be used in a SQL Transformation project.

The function type, for example, Numeric or String, is indicated in brackets next to the function name.

Some of the results in the examples have been rounded or truncated.

The actual values of database object IDs, such as the object ID of a table or the column ID of a column, might differ from the values shown in the examples.

ASCII Function [String]

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

```
ASCII ( string-expression )
```

Parameters

Table 48. Parameters

Parameter	Description
string-expression	The string.

Returns

SMALLINT

Example

The following statement returns the value 90, when the collation sequence is set to the default ISO_BINENG:

```
SELECT ASCII( 'Z' ) FROM iq_dummy
```

Usage

If the string is empty, **ASCII** returns zero. Literal strings must be enclosed in quotes.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

BYTE_SUBSTR64 and BYTE_SUBSTR Functions

BYTE_SUBSTR64 and **BYTE_SUBSTR** return the long binary byte substring of the `LONG BINARY` column parameter.

Usage

The **BYTE_SUBSTR64** and **BYTE_SUBSTR** functions also support the `LONG VARCHAR` data type and `LONG BINARY` and `LONG VARCHAR` variables of any data size.

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

CHAR function [String]

Returns the character with the ASCII value of a number.

Syntax

```
CHAR ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Returns

VARCHAR

Examples

The following statement returns the value “Y”:

```
SELECT CHAR( 89 ) FROM iq_dummy
```

The following statement returns the value “S”:

```
SELECT CHAR( 83 ) FROM iq_dummy
```

Usage

The character in the current database character set corresponding to the supplied numeric expression modulo 256 is returned.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

CHAR_LENGTH Function [String]

Returns the number of characters in a string.

Syntax

```
CHAR_LENGTH ( string-expression )
```

Parameters

Table 49. Parameters

Parameter	Description
string-expression	The string whose length is to be calculated.

Returns

INT

Usage

Trailing white space characters are included in the length returned.

The return value of a `NULL` string is NULL.

If the string is in a multibyte character set, the **CHAR_LENGTH** value may be less than the **BYTE_LENGTH** value.

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

CHAR_LENGTH64 Function

The **CHAR_LENGTH64** function returns an unsigned 64-bit value containing the character length of the `LONG VARCHAR` column parameter, including the trailing blanks.

Usage

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *Unstructured Data Analytics in Sybase IQ > Function Support*.

CHARINDEX Function [String]

Returns the position of the first occurrence of a specified string in another string.

Syntax

CHARINDEX (*string-expression1, string-expression2*)

Parameters

Parameter	Description
string-expression1	The string for which you are searching. This string is limited to 255 bytes.
string-expression2	The string to be searched. The position of the first character in the string being searched is 1.

Returns

INT

Example

The statement:

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX('K', Surname ) = 1
```

returns the following values:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

Usage

All the positions or offsets, returned or specified, in the **CHARINDEX** function are always character offsets and may be different from the byte offset for multibyte data.

If the string being searched contains more than one instance of the specified string, **CHARINDEX** returns the position of the first instance.

If the string being searched does not contain the specified string, **CHARINDEX** returns zero (0).

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is NULL.

CHARINDEX returns a 32 bit signed integer position for `CHAR` and `VARCHAR` columns.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DIFFERENCE Function [String]

Compares two strings, evaluates the similarity between them, and returns a value from 0 to 4.

The best match is 4.

Syntax

```
DIFFERENCE ( string-expression1, string-expression2 )
```

Parameters

Table 50. Parameters

Parameter	Description
string-expression1	The first string to compare.

Parameter	Description
string-expression2	The second string to compare.

Returns

SMALLINT

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DIFFERENCE Function Examples

Use the examples as reference for **DIFFERENCE** function usage.

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smith' ) FROM iq_dummy
```

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smyth' ) FROM iq_dummy
```

The following statement returns the value 3:

```
SELECT DIFFERENCE( 'Smith', 'Sweeney' ) FROM iq_dummy
```

The following statement returns the value 2:

```
SELECT DIFFERENCE( 'Smith', 'Jones' ) FROM iq_dummy
```

The following statement returns the value 1:

```
SELECT DIFFERENCE( 'Smith', 'Rubin' ) FROM iq_dummy
```

The following statement returns the value 0:

```
SELECT DIFFERENCE( 'Smith', 'Wilkins' ) FROM iq_dummy
```

GRAPHICAL_PLAN Function [String]

Returns the graphical query plan to Interactive SQL in an XML format string.

Syntax

```
GRAPHICAL_PLAN ( string-expression
[, statistics-level
[, cursor-type
[, update-status ]]])
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
string-expression	SQL statement for which the plan is to be generated. string-expression is generally a **SELECT** statement, but it can also be an **UPDATE** or **DELETE**, **INSERT SELECT**, or **SELECT INTO** statement.
statistics-level	An integer. Statistics-level can be: • 0 – Optimizer estimates only (default). • 2 – Detailed statistics including node statistics. • 3 – Detailed statistics.
cursor-type	A cursor type, expressed as a string. Possible values are: asensitive, insensitive, sensitive, or keyset-driven. If cursor-type is not specified, asensitive is used by default.
update-status	A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor: READ-ONLY – The cursor is read-only. READ-WRITE (default) – The cursor can be read or written to. READ-WRITE (default) – The cursor can be read or written to.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **GRAPHICAL_PLAN** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **GRAPHICAL_PLAN** to the correct data type and size.

Usage

If you do not provide an argument to the **GRAPHICAL_PLAN** function, the query plan is returned to you from the cache. If there is no query plan in the cache, then this message appears:

```
plan not available
```

The behavior of **GRAPHICAL_PLAN** function is controlled by database options `QUERY_PLAN_TEXT_ACCESS` and `QUERY_PLAN_TEXT_CACHING`. If `QUERY_PLAN_TEXT_ACCESS` is OFF (the default), then this message appears:

```
Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF
```

If a user needs access to the plan, the DBA must set option `QUERY_PLAN_TEXT_ACCESS` ON for that user.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_TEXT_ACCESS Option*. See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_TEXT_CACHING Option*.

If `QUERY_PLAN_TEXT_ACCESS` is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

If the query plan is not available in the cache and you are authorized to view plans on the client, then a query plan with optimizer estimates (query plan with `NOEXEC` option ON) is generated and appears on the Interactive SQL client plan window.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > NOEXEC Option*.

When a user requests a query plan that has not yet been executed, the query plan is not available in the cache. Instead, a query plan with optimizer estimates is returned without `QUERY_PLAN_AFTER_RUN` statistics.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_AFTER_RUN Option*.

Query plans for stored procedures are not accessible using the **GRAPHICAL_PLAN** function.

Users can view the query plan for cursors opened for Sybase IQ queries. A cursor is declared and opened using **DECLARE CURSOR** and **OPEN CURSOR**. To obtain the query plan for the most recently opened cursor, use:

```
SELECT GRAPHICAL_PLAN ( );
```

With the `QUERY_PLAN_AFTER_RUN` option OFF, the plan appears after **OPEN CURSOR** or **CLOSE CURSOR**. However, if `QUERY_PLAN_AFTER_RUN` is ON, **CLOSE CURSOR** must be executed before you request the plan.

For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL > Viewing plans using the Interactive SQL Plan Viewer > Viewing graphical plans in Interactive SQL*.

Examples

The following example passes a **SELECT** statement as a string parameter and returns the plan for executing the query. It saves the plan in the file `gplan.xml`.

Note: If you use the **OUTPUT** statement's **HEXADECIMAL** clause set to **ASIS** to get formatted plan output, the values of characters are written without any escaping, even if the value contains control characters. **ASIS** is useful for text that contains formatting characters such as tabs or carriage returns.

```
SELECT GRAPHICAL_PLAN ('SELECT * FROM Employees');OUTPUT to 'C:
\gplan.xml' HEXADECIMAL ASIS quote '';
```

The following example returns the query plan from the cache, if available:

```
SELECT GRAPHICAL_PLAN ( );
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

HTML_PLAN Function [String]

Returns query plans in an HTML format string.

Syntax

```
HTML_PLAN ( string-expression )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
string-expression	SQL statement for which the plan is to be generated. It is primarily a **SELECT** statement but can be an **UPDATE** or **DELETE** statement.

If you do not provide an argument to the **HTML_PLAN** function, the query plan is returned to you from the cache. If there is no query plan in the cache, this message appears:

```
No plan available
```

The behavior of the **HTML_PLAN** function is controlled by database options `QUERY_PLAN_TEXT_ACCESS` and `QUERY_PLAN_TEXT_CACHING`. If `QUERY_PLAN_TEXT_ACCESS` is OFF (the default), this message appears:

```
Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF
```

If `QUERY_PLAN_TEXT_ACCESS` is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_TEXT_ACCESS Option*. See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_TEXT_CACHING Option*.

The **HTML_PLAN** function can be used to return query plans to Interactive SQL using **SELECT**, **UPDATE**, **DELETE**, **INSERT SELECT**, and **SELECT INTO**.

Users can view the query plan for cursors opened for Sybase IQ queries. To obtain the query plan for the most recently opened cursor, use:

```
SELECT HTML_PLAN ( );
```

With `QUERY_PLAN_AFTER_RUN` option OFF, the plan appears after **OPEN CURSOR** or **CLOSE CURSOR**. However, if `QUERY_PLAN_AFTER_RUN` is ON, **CLOSE CURSOR** must be executed before you request the plan.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > QUERY_PLAN_AFTER_RUN Option*.

For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL > Viewing plans using the Interactive SQL Plan Viewer* .

When you request an **HTML_PLAN** for a SQL Anywhere query or for an OMNI/CIS decomposed query, the following message is returned:

```
No plan. HTML_PLAN function is not supported for this type of
statement or database.
```

Examples

The following example passes a **SELECT** statement as a string parameter and returns the HTML plan for executing the query. It saves the plan in the file `hplan.html`.

```
SELECT HTML_PLAN ('SELECT * FROM Employees');
OUTPUT to 'C:\hplan.html' HEXADECIMAL ASIS QUOTE '';
```

The **OUTPUT TO** clause **HEXADECIMAL ASIS** is useful for text that contains formatting characters such as tabs or carriage returns. When set to **ASIS**, values are written as is, without any escaping, even if the values contain control characters.

The following example returns the HTML query plan from the cache, if available.

```
SELECT HTML_PLAN ( );
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

INSERTSTR Function [String]

Inserts a string into another string at a specified position.

Syntax

```
INSERTSTR ( numeric-expression, string-expression1, string-
expression2 )
```

Parameters

Table 51. Parameters

Parameter	Definition
numeric-expression	The position after which *string-expression2* is to be inserted. Use zero to insert a string at the beginning.
string-expression1	The string into which *string-expression2* is to be inserted.
string-expression2	The string to be inserted.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **INSERTSTR** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **INSERTSTR** to the correct data type and size.

Example

The following statement returns the value “backoffice”:

```
SELECT INSERTSTR( 0, 'office ', 'back' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise. The STUFF function is equivalent and is supported in both Adaptive Server Enterprise and Sybase IQ.

LCASE Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LCASE ( string-expression )
```

Parameters

Table 52. Parameters

Parameter	Description
string-expression	The string to be converted to lowercase.

Returns

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LCASE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LCASE** to the correct data type and size.

Example

The following statement returns the value “lower case”:

```
SELECT LCASE( 'LOWER CasE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**LCASE** is not supported in Adaptive Server Enterprise; you can use **LOWER** to get the same functionality.

LEN Function [String]

Takes one argument as an input of type `BINARY` or `STRING` and returns the number of characters, as defined by the database's collation sequence, of a specified string expression, excluding trailing blanks.

The result may differ from the string’s byte length for multi-byte character sets.

`BINARY` and `VARBINARY` are also allowed, in which case LEN() returns the number of bytes of the input.

LEN is an alias of **LENGTH** function

Syntax

```
LEN ( string_expr )
```

Parameters

Table 53. Parameters

Parameters	Description
string_expr	The string expression to be evaluated.

Example

The following example returns the value 3152:

```
select len(Photo) from Productswhere ID = 500
```

Usage

This function is the equivalent of **CHAR_LENGTH** (*string_expression*).

Permissions

Any user can execute **LEN**.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.

LENGTH Function [String]

Returns the number of characters in the specified string.

Syntax

```
LENGTH ( string-expression )
```

Parameters

Table 54. Parameters

Parameter	Description
string-expression	The string.

Returns

INT

Example

The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, **LENGTH** returns the number of characters, not the number of bytes. If the string is of `BINARY` data type, the **LENGTH** function behaves as **BYTE_LENGTH**.

The **LENGTH** function is the same as the **CHAR_LENGTH** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use the **CHAR_LENGTH** function instead.

LOWER Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LOWER ( string-expression )
```

Parameters

Table 55. Parameters

Parameter	Description
string-expression	The string to be converted.

Returns

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LOWER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LOWER** to the correct data type and size.

Example

The following statement returns the value “lower case”:

```
SELECT LOWER( 'LOWER CasE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

LTRIM Function [String]

Removes leading blanks from a string.

Syntax

```
LTRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LTRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LTRIM** to the correct data type and size.

Example

The following statement returns the value “Test Message” with all leading blanks removed:

```
SELECT LTRIM( '     Test Message' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

OCTET_LENGTH Function [String]

Returns an unsigned 64-bit value containing the byte length of the column parameter.

Syntax

```
OCTET_LENGTH( column-name )
```

Parameters

Parameter	Description
column-name	The name of a column.

Usage

The return value of a NULL argument is NULL.

The **OCTET_LENGTH** function supports all Sybase IQ data types.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- Sybase—Not supported by SQL Anywhere or Adaptive Server Enterprise.

PATINDEX Function [String]

Returns the starting position of the first occurrence of a specified pattern.

Syntax

```
PATINDEX ( '%pattern%', string-expression )
```

Parameters

Parameter	Description
pattern	The pattern for which you are searching. This string is limited to 126 bytes for patterns with wildcards. If the leading percent wildcard is omitted, **PATINDEX** returns one (1) if the pattern occurs at the beginning of the string, and zero if not. If *pattern* starts with a percent wildcard, then the two leading percent wildcards are treated as one. Patterns without wildcards (percent % or under-score _) can be up to 255 bytes in length.
string-expression	The string to be searched for the pattern.

Returns

INT

Examples

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' ) FROM iq_dummy
```

The following statement returns the value 11:

```
SELECT PATINDEX ('%4_5_', '0a1A 2a3A 4a5A') FROM iq_dummy
```

Usage

PATINDEX returns the starting position of the first occurrence of the pattern. If the string being searched contains more than one instance of the string pattern, **PATINDEX** returns only the position of the first instance.

The pattern uses the same wildcards as the **LIKE** comparison. This table lists the pattern wildcards.

Table 56. PATINDEX pattern wildcards

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set

Wildcard	Matches
[^]	Any single character not in the specified range or set

If the pattern is not found, **PATINDEX** returns zero (0).

Searching for a pattern longer than 126 bytes returns NULL.

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is zero (0).

All the positions or offsets, returned or specified, in the **PATINDEX** function are always character offsets and may be different from the byte offset for multibyte data.

PATINDEX returns a 32 bit unsigned integer position for `CHAR` and `VARCHAR` columns.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

REPEAT Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPEAT ( string-expression, integer-expression )
```

Parameters

Parameter	Description
string-expression	The string to be repeated.
integer-expression	The number of times the string is to be repeated. If *integer-expression* is negative, an empty string is returned.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPEAT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set REPEAT to the correct data type and size.

Example

The following statement returns the value "repeatrepeatrepeat:"

```
SELECT REPEAT( 'repeat', 3 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise, but **REPLICATE** provides the same capabilities.

REPLACE Function [String]

Replaces all occurrences of a substring with another substring.

Syntax

REPLACE (*original-string*, *search-string*, *replace-string*)

Parameters

If any argument is NULL, the function returns NULL.

Parameter	Description
original-string	The string to be searched. This string can be any length.
search-string	The string to be searched for and replaced with *replace-string*. This string is limited to 255 bytes. If *search-string* is an empty string, the original string is returned unchanged.
replace-string	The replacement string, which replaces *search-string*. This can be any length. If *replace-string* is an empty string, all occurrences of *search-string* are deleted.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPLACE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REPLACE** to the correct data type and size.

Examples

The following statement returns the value "xx.def.xx.ghi:"

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' ) FROM iq_dummy
```

The following statement generates a result set containing **ALTER PROCEDURE** statements which, when executed, repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE(
  replace(proc_defn,'OldTableName','NewTableName'),
  'create procedure',
  'alter procedure')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%'
```

Use a separator other than the comma for the **LIST** function:

```
SELECT REPLACE( list( table_id ), ',', '--')
FROM  SYS.ISYSTAB
WHERE table_id <= 5
```

Usage

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

There are two ways to work around this issue:

- Declare a local temporary table, then perform an **INSERT**:

  ```
  DECLARE local temporary table #mytable
    (name_column char(10)) on commit preserve rows;
  INSERT INTO #mytable SELECT REPLACE(name,'0','1')   FROM
  dummy_table01;
  ```
- Use **CAST**:

  ```
  SELECT CAST(replace(name, '0', '1') AS Char(10)) into #mytable
  from dummy_table01;
  ```

If you need to control the width of the resulting column when *replace-string* is wider than *search-string*, use the **CAST** function. For example,

```
CREATE TABLE aa(a CHAR(5));
INSERT INTO aa VALUES(‘CCCCC’);
COMMIT;
SELECT a, CAST(REPLACE(a,’C’,’ZZ’) AS CHAR(5)) FROM aa;
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

REPLICATE Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPLICATE ( string-expression, integer-expression )
```

Parameters

Parameter	Description
string-expression	The string to be repeated.
integer-expression	The number of times the string is to be repeated.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPLICATE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REPLICATE** to the correct data type and size.

Example

The following statement returns the value “repeatrepeatrepeat:”

```
SELECT REPLICATE( 'repeat', 3 ) FROM iq_dummy
```

Usage

REPLICATE is the same as the **REPEAT** function.

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

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

REVERSE Function [String]

Takes one argument as an input of type `BINARY` or `STRING` and returns the specified string with characters listed in reverse order.

Syntax

REVERSE (*expression* | *uchar_expr*)

Parameters

Parameter	Description
expression	A character or binary-type column name, variable, or constant expression of `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, `BINARY`, or `VARBINARY` type.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REVERSE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REVERSE** to the correct data type and size.

Example 1

```
select reverse("abcd")
----
dcba
```

Example 2

```
select reverse(0x12345000)
----------
0x00503412
```

Usage

- **REVERSE**, a string function, returns the reverse of expression.
- If expression is NULL, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

Permissions

Any user can execute **REVERSE**.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.

RIGHT Function [String]

Returns the rightmost characters of a string.

Syntax

```
RIGHT ( string-expression, numeric-expression )
```

Parameters

Parameter	Description
string-expression	The string to be left-truncated.
numeric-expression	The number of characters at the end of the string to return.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **RIGHT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **RIGHT** to the correct data type and size.

Example

The following statement returns the value “olate:”

```
SELECT RIGHT( 'chocolate', 5 ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned might be greater than the specified number of characters.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

RTRIM Function [String]

Returns a string with trailing blanks removed.

Syntax

```
RTRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **RTRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **RTRIM** to the correct data type and size.

Example

The following statement returns the string “Test Message” with all trailing blanks removed.

```
SELECT RTRIM( 'Test Message     ' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SIMILAR Function [String]

Returns an integer between 0 and 100 representing the similarity between two strings.

Syntax

```
SIMILAR ( string-expression1, string-expression2 )
```

Parameters

Parameter	Description
string-expression1	The first string to be compared.

Parameter	Description
string-expression2	The second string to be compared.

Returns

SMALLINT

Example

The following statement returns the value 75:

```
SELECT SIMILAR( 'toast', 'coast' ) FROM iq_dummy
```

This signifies that the two values are 75% similar.

Usage

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers might have been added to the list more than once with slightly different names. Join the table to itself and produce a report of all similarities greater than 90 percent but less than 100 percent.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SORTKEY Function [String]

Generates values that can be used to sort character strings based on alternate collation rules.

Syntax

```
SORTKEY ( string-expression
[, { collation-id
| collation-name [(collation-tailoring-string)] } ]
)
```

Parameters

Parameter	Description
string-expression	The string expression must contain characters that are encoded in the character set of the database and must be STRING data type. If string-expression is NULL, the SORTKEY function returns a null value. An empty string has a different sort-order value than a null string from a database column. There is no limit on the length of the input string that the SORTKEY function can handle. The result of SORTKEY is always limited to 1024 bytes and is VARBINARY data type. If the actual results exceed 1024 bytes, the result contains only the first 1024 bytes.
collation-name	A string or character variable that specifies the name of the sort order to use. You can also specify the alias char_collation, or, equivalently, db_collation, to generate sort-keys as used by the CHAR collation in use by the database. Similarly, you can specify the alias NCHAR_COLLATION to generate sort-keys as used by the NCHAR collation in use by the database. However, Sybase IQ does not support NCHAR_COLLATION for Sybase IQ-specific objects. NCHAR_COLLATION is supported for SQL Anywhere objects on a Sybase IQ server.
collation-id	A variable, integer constant, or string that specifies the ID number of the sort order to use. This parameter applies only to Adaptive Server Enterprise collations, which can be referred to by their corresponding collation ID.

Parameter	Description
collation-tailoring-string	(Optional) Specify collation tailoring options (*collation-tailoring-string*) for additional control over sorting and comparison of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example, `‘UCA(locale=es;case=LowerFirst;accent=respect)’` The syntax for specifying these options is identical to the **COLLATION** clause of the **CREATE DATABASE** statement. . See *Reference: Statements and Options > SQL Statements > CREATE DATABASE Statement*. **Note:** All of the collation tailoring options are supported for SQL Anywhere databases, when specifying the Unicode Collation Algorithm (UCA) collation. For all other collations, only case sensitivity tailoring is supported.

Returns

BINARY

Example

The following statement queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sort-key values for the Surname column using the dict collation (Latin-1, English, French, German dictionary):

```
SELECT Surname, GivenName FROM Employees ORDER BY SORTKEY( Surname,
'dict' );
```

Usage

The **SORTKEY** function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the **SORTKEY** function.

For example, you can store the values returned by the **SORTKEY** function in a column with the source character string. The following **SELECT** statement retrieves data from table T1 in the sorted order of c1 according to the Thai dictionary:

```
SELECT rid, c1 from T1 ORDER BY SORTKEY(c1)
```

You instead store the value returned by **SORTKEY** in a column with the source character string. To retrieve the character data in the required order, the **SELECT** statement needs to include only an **ORDER BY** clause on the column that contains the results of running the **SORTKEY** function:

```
UPDATE T1 SET shadowc1=SORTKEY(c1) FROM T1;
SELECT rid, c1 FROM T1 ORDER BY shadowc1
```

The **SORTKEY** function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on `VARBINARY` data types.

Generating sort-keys for queries can be expensive. As an alternative for frequently requested sort-keys, consider creating a computed column to hold the sort-key values, and then referencing that column in the **ORDER BY** clause of the query.

If you do not specify a collation name or collation ID, the default is Default Unicode multilingual.

Valid collations are as follows:

- To see collations that are supported by Sybase IQ, listed by label, execute `iqinit -l`.
- The Adaptive Server Enterprise collations are listed in the table below.

Description	Collation name	Collation ID
Default Unicode multilingual	default	0
CP 850 Alternative: no accent	altnoacc	39
CP 850 Alternative: lowercase first	altdict	45
CP 850 Western European: no case, preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian: no case, preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case, preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57

Description	Collation name	Collation ID
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-5 Turkish dictionary	turdict	72
ISO 8859-5 Turkish no accents	turnoac	73
ISO 8859-5 Turkish no case	turnocs	74
CP 874 (TIS 620) Royal Thai dictionary	thaidict	1
ISO 14651 ordering standard	14651	22
Shift-JIS binary order	sjisbin	179
Unicode UTF-8 binary sort	utf8bin	24
EUC JIS binary order	eucjisbn	192
GB2312 binary order	gb2312bn	137
CP932 MS binary order	cp932bin	129
Big5 binary order	big5bin	194
EUC KSC binary order	euckscbn	161

With respect to collation tailoring, full sensitivity is generally the intent when creating sort-keys, so when you specify a non-UCA collation, the default tailoring applied is equivalent to case=Respect. For example, the following two statements are equivalent:

```
SELECT SORTKEY( 'abc', '1252LATIN1' );
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

When specifying a non-UCA collation, by default, collation tailorings are accent and case-sensitive. However, for non-UCA collations, you can override only the case sensitivity using a collation tailoring. For example:

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=LowerFirst)' );
```

If the database was created without specifying tailoring options, the following two clauses may generate different sort orders, even if the database collation name is specified for the **SORTKEY** function:

```
ORDER BY string-expression
```

```
ORDER BY SORTKEY( string-expression, database-collation-name )
```

Different sort orders may be generated, because the default tailoring settings used for database creation and for the **SORTKEY** function are different. To get the same behavior from **SORTKEY** as for the database collation, either provide a tailoring syntax for *collation-tailoring-string* that matches the settings for the database collation, or specify `db_collation` for collation-name. For example:

```
SORTKEY( expression, 'db_collation' )
```

Note: Sort-key values created using a version of Sybase IQ earlier than 15.0 do not contain the same values created using version 15.0 and later. This may be a problem for your applications if your pre-15.0 database has sort-key values stored within it, especially if sort-key value comparison is required by your application. Regenerate any sort-key values in your database that were generated using a version of Sybase IQ earlier than 15.0.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

SOUNDEX Function [String]

Returns a number representing the sound of a string.

Syntax

```
SOUNDEX ( string-expression )
```

Parameters

Parameters	Description
string-expression	The string.

Returns

SMALLINT

Example

The following statement returns two numbers, representing the sound of each name. The **SOUNDEX** value for each argument is 3827.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' ) FROM iq_dummy
```

SOUNDEX ('Smith') is equal to **SOUNDEX** ('Smythe').

Usage

The **SOUNDEX** function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Doubled letters are counted as one letter. For example:

```
SOUNDEX( 'apples' ) FROM iq_dummy
```

is based on the letters A, P, L, and S.

Multibyte characters are ignored by the **SOUNDEX** function.

Although it is not perfect, **SOUNDEX** normally returns the same number for words that sound similar and that start with the same letter.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise, except that Adaptive Server Enterprise returns a CHAR(4) result and Sybase IQ returns an integer.

SPACE Function [String]

Returns a specified number of spaces.

Syntax

```
SPACE ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The number of spaces to return.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SPACE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SPACE** to the correct data type and size.

Example

The following statement returns a string containing 10 spaces:

```
SELECT SPACE( 10 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Compatible with Adaptive Server Enterprise.

STR Function [String]

Returns the string equivalent of a number.

Syntax

```
STR ( numeric-expression [ , length [ , decimal ] ] )
```

Parameters

Parameter	Description
numeric-expression	Any approximate numeric (FLOAT, REAL, or DOUBLE precision) expression.
length	The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, the sign, if any, and blanks). The default is 10 and the maximum length is 255.
decimal	The number of digits to the right of the decimal point to be returned. The default is 0.

Returns

VARCHAR

Examples

The following statement returns a string of six spaces followed by 1234, for a total of ten characters:

```
SELECT STR( 1234.56 ) FROM iq_dummy
```

The following statement returns the result 1234.5:

```
SELECT STR( 1234.56, 6, 1 ) FROM iq_dummy
```

Usage

If the integer portion of the number cannot fit in the length specified, then the result is NULL. For example, the following statement returns NULL:

```
SELECT STR( 1234.56, 3 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

STRING Function [String]

Concatenates one or more strings into one large string.

Syntax

```
STRING ( string-expression [ , … ] )
```

Parameters

Parameter	Description
string-expression	A string. If only one argument is supplied, it is converted into a single expression. If more than one argument is supplied, they are concatenated into a single string. A NULL is treated as an empty string ('').

Returns

LONG VARCHAR

LONG NVARCHAR

LONG BINARY

Note: The result data type is a `LONG VARCHAR`. If you use **STRING** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **STRING** to the correct data type and size.

Example

The following statement returns the value testing123:

```
SELECT STRING( 'testing', NULL, 123 )
FROM iq_dummy
```

Usage

Numeric or date parameters are converted to strings before concatenation. You can also use the **STRING** function to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, **STRING** returns NULL.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

STRTOUUID Function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

Syntax

```
STRTOUUID ( string-expression )
```

Parameters

Parameter	Description
string-expression	A string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*

Returns

UNIQUEIDENTIFIER

Example

```
CREATE TABLE T (
 pk uniqueidentifier primary key,
 c1 int);
INSERT INTO T (pk, c1)
 VALUES (STRTOUUID
 ('12345678-1234-5678-9012-123456789012'), 1);
```

Usage

Converts a string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* where *x* is a hexadecimal digit, to a unique identifier value. If the string is not a valid UUID string, NULL is returned.

You can use **STRTOUUID** to insert UUID values into a Sybase IQ database.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

STUFF Function [String]

Deletes a number of characters from one string and replaces them with another string.

Syntax

```
STUFF ( string-expression1, start, length, string-expression2 )
```

Parameters

Parameter	Description
string-expression1	The string to be modified by the **STUFF** function.
start	The character position at which to begin deleting characters. The first character in the string is position 1.
length	The number of characters to delete.
string-expression2	The string to be inserted. To delete a portion of a string using the **STUFF** function, use a replacement string of NULL

Returns

LONG NVARCHAR

Example

The following statement returns the value “chocolate pie”:

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' )
FROM iq_dummy
```

Usage

To delete a portion of a string using **STUFF**, use a replacement string of NULL. To insert a string using **STUFF**, use a length of zero.

The **STUFF** function will return a NULL result in the following situations:

- Any of the first three parameters is a NULL value.
- Either the start or length parameter is a negative value.
- The start parameter is greater than the length of string-expression1.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SUBSTRING Function [String]

Returns a substring of a string.

Syntax

```
{ SUBSTRING | SUBSTR } ( string-expression, start [ , length ] )
```

Parameters

Parameter	Description
string-expression	The string from which a substring is to be returned.
start	The start position of the substring to return, in characters. A negative starting position specifies a number of characters from the end of the string instead of the beginning. The first character in the string is at position 1.
length	The length of the substring to return, in characters. A positive *length* specifies that the substring ends *length* characters to the right of the starting position, while a negative *length* specifies that the substring ends *length* characters to the left of the starting position.

Returns

LONG VARCHAR

LONG NVARCHAR

LONG BINARY

Note: The result data type is a `LONG VARCHAR`. If you use **STRING** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **STRING** to the correct data type and size.

Examples

The following statement returns “back”:

```
SELECT SUBSTRING ( 'back yard', 1 , 4 )
FROM iq_dummy
```

The following statement returns yard:

```
SELECT SUBSTR ( 'back yard', -1 , -4 )
FROM iq_dummy
```

The following statement returns 0x2233:

```
SELECT SUBSTR ( 0x112233445566, 2, 2 )
FROM iq_dummy
```

Usage

If *length* is specified, the substring is restricted to that length. If no length is specified, the remainder of the string is returned, starting at the *start* position.

Both *start* and *length* can be negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**SUBSTR** is not supported by Adaptive Server Enterprise. Use **SUBSTRING** instead

SUBSTRING64 Function [String]

The **SUBSTRING64** function returns a variable-length character string of the large object column or variable parameter.

Usage

SUBSTRING64 supports searching `LONG VARCHAR` and `LONG BINARY` columns and `LONG VARCHAR` and `LONG BINARY` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

TRIM Function [String]

Removes leading and trailing blanks from a string.

Syntax

```
TRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **TRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **TRIM** to the correct data type and size.

Example

The following statement returns the value "chocolate" with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use **LTRIM** and **RTRIM** instead

UCASE Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UCASE ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be converted to uppercase.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **UCASE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **UCASE** to the correct data type and size.

Example

The following statement returns the value "CHOCOLATE":

```
SELECT UCASE( 'ChocoLate' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**UCASE** is not supported by Adaptive Server Enterprise, but **UPPER** provides the same feature in a compatible manner

UPPER Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UPPER ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be converted to uppercase.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **UPPER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **UPPER** to the correct data type and size.

Example

The following statement returns the value “CHOCOLATE”:

```
SELECT UPPER( 'ChocoLate' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

UUIDTOSTR Function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

Syntax

```
UUIDTOSTR ( uuid-expression )
```

Parameters

Table 57. Parameters

Parameter	Description
uuid-expression	A unique identifier value.

Returns

VARCHAR

Example

To convert a unique identifier value into a readable format, execute a query similar to:

```
CREATE TABLE T3 (
pk uniqueidentifier primary key,c1 int);
INSERT INTO T3 (pk, c1)
values (0x12345678123456789012123456789012, 1)
SELECT UUIDTOSTR(pk) FROM T3
```

Usage

Converts a unique identifier to a string value in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*, where x is a hexadecimal digit. If the binary value is not a valid unique identifier, NULL is returned.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

HTML_DECODE Function [HTTP]

Decodes special character entities that appear in HTML literal strings.

Syntax

```
HTML_DECODE ( string )
```

Note: CIS functional compensation performance considerations apply.

The **HTML_DECODE** function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTML_DECODE function [HTTP]*.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **HTML_DECODE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **HTML_DECODE** to the correct data type and size.

HTML_ENCODE Function [HTTP]

Encodes special characters within strings to be inserted into HTML documents.

Syntax

```
HTML_ENCODE ( string )
```

Note: CIS functional compensation performance considerations apply.

The **HTML_ENCODE** function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTML_ENCODE function [HTTP].*

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **HTML_ENCODE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **HTML_ENCODE** to the correct data type and size.

HTTP_DECODE Function [HTTP]

Decodes special characters within strings for use with HTTP.

Syntax

```
HTTP_DECODE ( string )
```

Note: CIS functional compensation performance considerations apply.

The `HTTP_DECODE` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTTP_DECODE function [HTTP].*

HTTP_ENCODE Function [HTTP]

Encodes special characters in strings for use with HTTP.

Syntax

```
HTTP_ENCODE ( string )
```

Note: CIS functional compensation performance considerations apply.

The `HTTP_ENCODE` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTTP_ENCODE function [HTTP].*

HTTP_HEADER Function [HTTP]

Gets the value of an HTTP header.

Syntax

```
HTTP_HEADER ( field-name )
```

Note: CIS functional compensation performance considerations apply.

The `HTTP_HEADER` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTTP_HEADER function [HTTP].*

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **HTTP_HEADER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **HTTP_HEADER** to the correct data type and size.

HTTP_VARIABLE function [HTTP]

Gets the value of an HTTP variable.

Syntax

```
HTTP_VARIABLE ( var-name [ [ , instance ] , header-field )
```

Note: CIS functional compensation performance considerations apply.

The `HTTP_VARIABLE` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > HTTP_VARIABLE function [HTTP].*

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **HTTP_VARIABLE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **HTTP_VARIABLE** to the correct data type and size.

NEXT_HTTP_HEADER Function [HTTP]

Gets the next HTTP header name.

Syntax

```
NEXT_HTTP_HEADER ( header-name )
```

Note: CIS functional compensation performance considerations apply.

The `NEXT_HTTP_HEADER` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > NEXT_HTTP_HEADER function [HTTP].*

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **NEXT_HTTP_HEADER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **NEXT_HTTP_HEADER** to the correct data type and size.

NEXT_HTTP_VARIABLE Function [HTTP]

Get the next HTTP variable name.

Syntax

```
NEXT_HTTP_VARIABLE ( var-name )
```

Note: CIS functional compensation performance considerations apply.

The `NEXT_HTTP_VARIABLE` function is a SQL Anywhere function. See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > Using SQL > SQL functions > SQL functions (E-O) > NEXT_HTTP_ VARIABLE function [HTTP].*

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **NEXT_HTTP_VARIABLE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **NEXT_HTTP_VARIABLE** to the correct data type and size.

ABS Function [Numeric]

Returns the absolute value of a numeric expression.

Syntax

```
ABS ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number whose absolute value is to be returned.

Returns

An absolute value of the numeric expression.

Numeric-expression data type	Returns
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

Example

The following statement returns the value 66:

```
SELECT ABS( -66 ) FROM iq_dummy
```

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ACOS Function [Numeric]

Returns the arc-cosine, in radians, of a numeric expression.

Syntax

```
ACOS ( numeric-expression )
```

Parameters

Table 58. Parameters

Parameter	Description
numeric-expression	The cosine of the angle.

Returns

DOUBLE

Example

The following statement returns the value 1.023945:

```
SELECT ACOS( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ASIN Function [Numeric]

Returns the arc-sine, in radians, of a number.

Syntax

```
ASIN ( numeric-expression )
```

Parameters

Table 59. Parameters

Parameter	Description
numeric-expression	The sine of the angle.

Returns

DOUBLE

Example

The following statement returns the value 0.546850.

```
SELECT ASIN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ATAN Function [Numeric]

Returns the arc-tangent, in radians, of a number.

Syntax

```
ATAN ( numeric-expression )
```

Parameters

Table 60. Parameters

Parameter	Description
numeric-expression	The tangent of the angle.

Returns

DOUBLE

Example

The following statement returns the value 0.479519:

```
SELECT ATAN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ATAN2 Function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers.

Syntax

```
ATAN2 ( numeric-expression1, numeric-expression2 )
```

Parameters

Table 61. Parameters

Parameter	Description
numeric-expression1	The numerator in the ratio whose arc tangent is calculated.
numeric-expression2	The denominator in the ratio whose arc-tangent is calculated.

Returns

DOUBLE

Example

The following statement returns the value 0.00866644968879073143:

```
SELECT ATAN2( 0.52, 060 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—ATAN2 is not supported by Adaptive Server Enterprise.

BYTE_LENGTH Function [String]

Returns the number of bytes in a string.

Syntax

```
BYTE_LENGTH ( string-expression )
```

Parameters

Table 62. Parameters

Parameters	Description
string-expression	The string whose length is to be calculated.

Returns

INT

Example

Returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' ) FROM iq_dummy
```

Usage

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the **BYTE_LENGTH** value differs from the number of characters returned by **CHAR_LENGTH**.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

BYTE_LENGTH64 Function

BYTE_LENGTH64 returns an unsigned 64-bit value containing the byte length of the `LONG BINARY` column parameter.

Usage

BYTE_LENGTH64 also supports the `LONG VARCHAR` data type and `LONG BINARY` and `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

CEIL Function [Numeric]

Returns the smallest integer greater than or equal to the specified expression.

CEIL is as synonym for **CEILING**.

Syntax

```
CEIL ( numeric-expression )
```

Parameters

Parameters	Description
expression	A column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, **CEIL** generates an error. The return value has the same data type as the value supplied.

Usage

For a given expression, the **CEIL** function takes one argument. For example, **CEIL (-123.45)** returns -123. **CEIL (123.45)** returns 124.

See *System Administration Guide: Volume 1 > International Language and Character Sets*.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

CEILING Function [Numeric]

Returns the ceiling (smallest integer not less than) of a number.

CEIL is as synonym for **CEILING**.

Syntax

```
CEILING ( numeric-expression )
```

Parameters

Table 63. Parameters

Parameter	Description
numeric-expression	The number whose ceiling is to be calculated.

Returns

DOUBLE

Examples

The following statement returns the value 60.00000:

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT CEILING( 123 ) FROM iq_dummy
```

The following statement returns the value 124.00:

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

The following statement returns the value -123.00:

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

COS Function [Numeric]

Returns the cosine of a number, expressed in radians.

Syntax

```
COS ( numeric-expression )
```

Parameters

Table 64. Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 0.86781:

```
SELECT COS( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

COT Function [Numeric]

Returns the cotangent of a number, expressed in radians.

Syntax

```
COT ( numeric-expression )
```

Parameters

Table 65. Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 1.74653:

```
SELECT COT( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DEGREES Function [Numeric]

Converts a number from radians to degrees.

Syntax

```
DEGREES ( numeric-expression )
```

Parameters

Table 66. Parameters

Parameter	Description
numeric-expression	An angle in radians.

Returns

Returns the degrees of the angle given by *numeric-expression*.

DOUBLE

Example

The following statement returns the value 29.793805:

```
SELECT DEGREES( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

EXP Function [Numeric]

Returns the exponential function, e to the power of a number.

Syntax

```
EXP ( numeric-expression )
```

Parameters

Table 67. Parameters

Parameter	Description
numeric-expression	The exponent.

Returns

DOUBLE

Example

The following statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

FLOOR Function [Numeric]

Returns the floor of (largest integer not greater than) a number.

Syntax

```
FLOOR ( numeric-expression )
```

Parameters

Table 68. Parameters

Parameter	Description
numeric-expression	The number, usually a float.

Returns

DOUBLE

Examples

The following statement returns the value 123.00:

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

The following statement returns the value -124.00.

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

LN Function [Numeric]

Returns the natural logarithm of the specified expression.

Syntax

```
LN ( numeric-expression )
```

Parameters

Parameter	Description
expression	Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the **LN** function generates an error. The return value is of `DOUBLE` data type.

Usage

LN takes one argument. For example, **LN** (*20*) returns 2.995732.

The **LN** function is an alias of the **LOG** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use the LOG function instead.

LOG Function [Numeric]

Returns the natural logarithm of a number.

LN is an alias of **LOG**.

Syntax

```
LOG ( numeric-expression )
```

Parameters

Table 69. Parameters

Parameter	Description
numeric-expression	The number.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 3.912023:

```
SELECT LOG( 50 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

LOG10 Function [Numeric]

Returns the base 10 logarithm of a number.

Syntax

```
LOG10 ( numeric-expression )
```

Parameters

Table 70. Parameters

Parameter	Description
numeric-expression	The number.

Returns

This function converts its argument to DOUBLE, and performs the computation in double-precision floating point. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 1.698970.

```
SELECT LOG10( 50 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

POWER Function [Numeric]

Calculates one number raised to the power of another.

Syntax

```
POWER ( numeric-expression1, numeric-expression2 )
```

Parameters

Parameter	Description
numeric-expression1	The base.
numeric-expression2	The exponent.

Returns

DOUBLE

Example

The following statement returns the value 64:

```
SELECT Power( 2, 6 ) FROM iq_dummy
```

Usage

Raises *numeric-expression1* to the power *numeric-expresson2*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

RADIANS Function [Numeric]

Converts a number from degrees to radians.

Syntax

```
RADIANS ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	A number, in degrees. This angle is converted to radians

Returns

DOUBLE

Example

The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

RAND Function [Numeric]

Returns a `DOUBLE` precision, random number x, where 0 <= x <1, with an optional seed.

Syntax

```
RAND ( [ integer-expression ] )
```

Parameters

Parameter	Description
integer-expression	The optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

Returns

DOUBLE

Examples

The following statement returns a 5% sampling of a table:

```
SELECT AVG(table1.number_of_cars), AVG(table1.number_of_tvs)FROM
table1 WHERE RAND(ROWID(table1)) < .05 and table1.income < 50000;
```

The following statement returns a value of approximately 941392926249216914:

```
SELECT RAND( 4 ) FROM iq_dummy
```

Usage

If **RAND** is called with a **FROM** clause and an argument in a query containing only tables in IQ stores, the function returns an arbitrary but repeatable value.

When no argument is called, **RAND** is a non-deterministic function. Successive calls to **RAND** might return different values. The query optimizer does not cache the results of the **RAND** function

Note: The values returned by **RAND** vary depending on whether you use a **FROM** clause or not and whether the referenced table was created in `SYSTEM` or in an IQ store.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

REMAINDER Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
REMAINDER ( dividend, divisor )
```

Parameters

Parameter	Description
dividend	The dividend, or numerator of the division.
divisor	The divisor, or denominator of the division.

Returns

INTEGER

NUMERIC

Example

The following statement returns the value 2:

```
SELECT REMAINDER( 5, 3 ) FROM iq_dummy
```

Usage

REMAINDER is the same as the **MOD** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported in Adaptive Server Enterprise. The % (modulo) operator and the division operator can be used to produce a remainder.

ROUND Function [Numeric]

Rounds the *numeric-expression* to the specified *integer-expression* number of places after the decimal point.

Syntax

```
ROUND ( numeric-expression, integer-expression )
```

Parameters

Parameter	Description
numeric-expression	The number, passed to the function, to be rounded.
integer-expression	A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC

Examples

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 ) FROM iq_dummy
```

Additional results of the **ROUND** function are shown in the following table:

Value	ROUND (Value)
123.4567	round (a.n,4)
123.4570	round (a.n,3)
123.4600	round (a.n,2)
123.5000	round (a.n,1)
123.0000	round (a.n, 0)
120.0000	round (a.n,-1)
100.0000	round (a.n,-2)

Value	ROUND (Value)
0.0000	round(a.n,-3)

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SIN Function [Numeric]

Returns the sine of a number, expressed in radians.

Syntax

```
SIN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

DOUBLE

Example

The following statement returns the value 0.496880:

```
SELECT SIN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SQUARE Function [Numeric]

Returns the square of the specified expression as a float.

Syntax

```
SQUARE ( numeric-expression )
```

Parameters

Parameter	Description
expression	Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the SQUARE function generates an error. The return value is of DOUBLE data type.

Usage

SQUARE function takes one argument. For example, **SQUARE** (*12.01*) returns 144.240100.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

TAN Function [Numeric]

Returns the tangent of a number.

Syntax

```
TAN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	An angle, in radians.

Returns

DOUBLE

Example

Returns the value 0.572561:

```
SELECT TAN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

TRUNCNUM Function [Numeric]

Truncates a number at a specified number of places after the decimal point.

Syntax

```
TRUNCNUM ( numeric-expression, integer-expression )
```

Parameters

Parameter	Description
numeric-expression	The number to be truncated.
integer-expression	A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC

Examples

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 ) FROM iq_dummy
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 ) FROM iq_dummy
```

Usage

This function is the same as **TRUNCATE**, but does not cause keyword conflicts.

You can use combinations of **ROUND**, **FLOOR**, and **CEILING** to provide similar functionality.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

WIDTH_BUCKET Function [Numerical]

For a given expression, the **WIDTH_BUCKET** function returns the bucket number that the result of this expression will be assigned after it is evaluated.

Syntax

```
WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )
```

Parameters

Parameter	Description
expression	The expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If *expr* evaluates to null, then the expression returns null.
min_value	An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.
max_value	An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.
num_buckets	Is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

Examples

The following example creates a ten-bucket histogram on the `credit_limit` column for customers in Massachusetts in the sample table and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```
select EmployeeID, Surname, Salary, WIDTH_BUCKET(Salary, 29000,
60000, 4) "Wages" from Employees where State = 'FL' order by "Wages"
```

```
EMPLOYEEID   SURNAME                SALARY        Wages
----------   -------                ------        -----
888          Charlton            28300.000            0
1390         Litton              58930.000            4
207          Francis             53870.000            4
266          Gowda               59840.000            4
445          Lull                87900.000            5
1021         Sterling            64900.000            5
902          Kelly               87500.000            5
1576         Evans               68940.000            5
```

When the bounds are reversed, the buckets are open-closed intervals. For example: **WIDTH_BUCKET** (*credit_limit*, *5000*, *0*, *5*). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

Usage

You can generate equiwidth histograms with the **WIDTH_BUCKET** function. Equiwidth histograms divide data sets into buckets whose interval size (highest value to lowest value) is equal. The number of rows held by each bucket will vary. A related function, **NTILE**, creates equiheight buckets.

Equiwidth histograms can be generated only for numeric, date or datetime data types; therefore, the first three parameters should be all numeric expressions or all date expressions. Other types of expressions are not allowed. If the first parameter is NULL, the result is NULL. If the second or the third parameter is NULL, an error message is returned, as a NULL value cannot denote any end point (or any point) for a range in a date or numeric value dimension. The last parameter (number of buckets) should be a numeric expression that evaluates to a positive integer value; 0, NULL, or a negative value will result in an error.

Buckets are numbered from 0 to (n+1). Bucket 0 holds the count of values less than the minimum. Bucket(n+1) holds the count of values greater than or equal to the maximum specified value.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

UDAF example: my_bit_xor declaration

The “my_bit_xor” example is analogous to the SQL Anywhere (SA) built-in BIT_XOR, except it operates only on unsigned integers.

my_bit_xor declaration

The resulting declaration is:

```
CREATE AGGREGATE FUNCTION my_bit_xor(IN arg1 UNSIGNED    INT)
 RETURNS UNSIGNED INT
    ON EMPTY INPUT RETURNS NULL
    EXTERNAL NAME 'describe_my_bit_xor@my_shared_lib'
```

Like the my_sum example, my_bit_xor has no associated usage restrictions, and is therefore usable as a simple aggregate or as an OLAP-style aggregate with any kind of a window.

UDAF example: my_sum definition

The "my_sum" example operates only on integers.

my_sum definition

Since my_sum, like SUM, can be used in any context, all the optimized optional entry points have been supplied. In this example, the normal _evaluate_extfn function can also be used as the _evaluate_superaggregate_extfn function.

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>


//  Simple aggregate UDF that adds up a set of
//  integer arguments, and whenever asked returns
//  the resulting big integer total.  For int
//  arguments, the only difference between this
//  UDF and the SUM built-in aggregate is that this
//  UDF will return NULL if there are no input rows.
//
//  The start function creates a little structure for
//  the running total, and the finish function then
//  deallocates it.
//
//  Since there are no aggregate usage restrictions
//  for this UDAF, the corresponding SQL declaration
//  will look like:
//
//              CREATE AGGREGATE FUNCTION my_sum(IN arg1 INT)
//                      RETURNS BIGINT
//                      ON EMPTY INPUT RETURNS NULL
//                      EXTERNAL NAME 'my_integer_sum@libudfex'


typedef struct my_total {
  a_sql_int64   _total;
  a_sql_uint64  _num_nonnulls_seen;
} my_total;


extern "C"
void my_integer_sum_start(a_v3_extfn_aggregate_context *cntxt)
{
}


extern "C"
void my_integer_sum_finish(a_v3_extfn_aggregate_context *cntxt)
{
}


extern "C"
void my_integer_sum_reset(a_v3_extfn_aggregate_context *cntxt)
{
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;
  cptr->_total = 0;
  cptr->_num_nonnulls_seen = 0;
}


extern "C"
void my_integer_sum_next_value(a_v3_extfn_aggregate_context *cntxt,
                               void *arg_handle)
```

```
{
  an_extfn_value  arg;
  a_sql_int32 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it to the total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }
}


extern "C"
void my_integer_sum_drop_value(a_v3_extfn_aggregate_context *cntxt,
                               void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_int32 arg1;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then subtract it from the
total
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total -= arg1;
    cptr->_num_nonnulls_seen--;
  }
}


extern "C"
void my_integer_sum_evaluate(a_v3_extfn_aggregate_context *cntxt,
                             void *arg_handle)
{
  an_extfn_value  outval;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Set the output result value.  If the inputs
  //  were all NULL, then set the result as NULL.
  //
  outval.type = DT_BIGINT;
  outval.piece_len = sizeof(a_sql_int64);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_total;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


extern "C"
```

```
void my_integer_sum_cum_evaluate(
                        a_v3_extfn_aggregate_context *cntxt,
                        void *arg_handle)
{
  an_extfn_value  outval;
  an_extfn_value  arg;
  int arg1;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it into the
total.
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }

  //  Then set the output result value.  If the inputs
  //  were all NULL, then set the result as NULL.
  //
  outval.type = DT_BIGINT;
  outval.piece_len = sizeof(a_sql_int64);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_total;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


extern "C"
void my_integer_sum_next_subagg_value(
                a_v3_extfn_aggregate_context *cntxt,
                void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_int64 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it to the total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int64 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }
}


extern "C"
void my_integer_sum_drop_subagg_value(
                a_v3_extfn_aggregate_context *cntxt,
                void *arg_handle)
```

```
{
  an_extfn_value  arg;
  a_sql_int64 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then subtract it from the
total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int64 *)arg.data);
    cptr->_total -= arg1;
    cptr->_num_nonnulls_seen--;
  }
}



a_v3_extfn_aggregate my_integer_sum_descriptor =
{
        &my_integer_sum_start,
        &my_integer_sum_finish,
        &my_integer_sum_reset,
        &my_integer_sum_next_value,
        &my_integer_sum_evaluate,
        &my_integer_sum_drop_value,
        &my_integer_sum_cum_evaluate,
        &my_integer_sum_next_subagg_value,
        &my_integer_sum_drop_subagg_value,
        &my_integer_sum_evaluate,
        NULL, // reserved1_must_be_null
        NULL, // reserved2_must_be_null
        NULL, // reserved3_must_be_null
        NULL, // reserved4_must_be_null
        NULL, // reserved5_must_be_null
        0, // indicators
        ( short )sizeof( my_total ), // context size
        8, // context alignment
        0.0, //external_bytes_per_group
        0.0, // external bytes per row
        0, // reserved6_must_be_null
        0, // reserved7_must_be_null
        0, // reserved8_must_be_null
        0, // reserved9_must_be_null
        0, // reserved10_must_be_null
        NULL // _for_server_internal_use
};

extern "C"
a_v3_extfn_aggregate *my_integer_sum()
{
  return &my_integer_sum_descriptor;
}
```

UDAF example: my_bit_or definition

The "my_bit_or" example is similar to the SA built-in BIT_OR, except my_bit_or operates only on unsigned integers, and can be used only as a simple aggregate.

my_bit_or definition

The "my_bit_or" definition is somewhat simpler than the "my_bit_xor" example.

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>


//  A simple (non-OLAP) aggregate UDF that ORs a set
//  of unsigned integer arguments, and whenever asked
//  returns the resulting unsigned integer result.
//
//  The start function creates a little structure for
//  the running result, and the finish function then
//  deallocates it.
//
//  The aggregate usage restrictions for this UDAF
//  only allow its use as a simple aggregate, so the
//  corresponding SQL declaration will look like:
//
//      CREATE AGGREGATE FUNCTION my_bit_or(IN arg1 UNSIGNED INT)
//            RETURNS UNSIGNED INT
//                      ON EMPTY INPUT RETURNS NULL
//            OVER NOT ALLOWED
//            EXTERNAL NAME 'my_bit_or@libudfex'


typedef struct my_or_result {
  a_sql_uint32 _or_result;
  a_sql_uint32 _non_null_seen;
} my_or_result;



#if defined __cplusplus
extern "C" {
#endif

static void my_or_start(a_v3_extfn_aggregate_context *cntxt)
{
}

static void my_or_finish(a_v3_extfn_aggregate_context *cntxt)
{
}

static void my_or_reset(a_v3_extfn_aggregate_context *cntxt)
{
  my_or_result *cptr = (my_or_result *)cntxt-
```

```
>_user_calculation_context;
  cptr->_or_result = 0;
  cptr->_non_null_seen = 0;
}


static void my_or_next_value(a_v3_extfn_aggregate_context *cntxt,
                             void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_uint32 arg1;

  my_or_result *cptr = (my_or_result *)cntxt-
>_user_calculation_context;

  //  Get the one argument, and add it to the total
  if (cntxt->get_value( arg_handle, 1, &arg ) && arg.data)
  {
      arg1 = *((a_sql_uint32 *)arg.data);
      cptr->_or_result |= arg1;
      cptr->_non_null_seen = 1;
  }
}


static void my_or_evaluate(a_v3_extfn_aggregate_context *cntxt,
                           void *arg_handle)
{
  an_extfn_value  outval;
  my_or_result *cptr = (my_or_result *)cntxt-
>_user_calculation_context;

  outval.type = DT_UNSINT;
  outval.piece_len = sizeof(a_sql_uint32);
  if (cptr->_non_null_seen)
  {
      outval.data = &cptr->_or_result;
  }
  else
  {
      // Return null if no values seen
      outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}



static a_v3_extfn_aggregate my_or_descriptor =
{
    &my_or_start,
    &my_or_finish,
    &my_or_reset,
    &my_or_next_value,
    &my_or_evaluate,
    NULL, // drop_val_extfn
    NULL, // cume_eval,
    NULL, // next_subaggregate_extfn
    NULL, // drop_subaggregate_extfn
    NULL, // evaluate_superaggregate_extfn
    NULL, // reserved1_must_be_null
    NULL, // reserved2_must_be_null
    NULL, // reserved3_must_be_null
    NULL, // reserved4_must_be_null
    NULL, // reserved5_must_be_null
    0, // indicators
    ( short )sizeof( my_or_result ), // context size
    8, // context alignment
    0.0, //external_bytes_per_group
    0.0, // external bytes per row
    0, // reserved6_must_be_null
    0, // reserved7_must_be_null
    0, // reserved8_must_be_null
    0, // reserved9_must_be_null
    0, // reserved10_must_be_null
    NULL // _for_server_internal_use
};


extern "C"
a_v3_extfn_aggregate *my_bit_or()
{
  return &my_or_descriptor;
}


#if defined __cplusplus
}
#endif
```

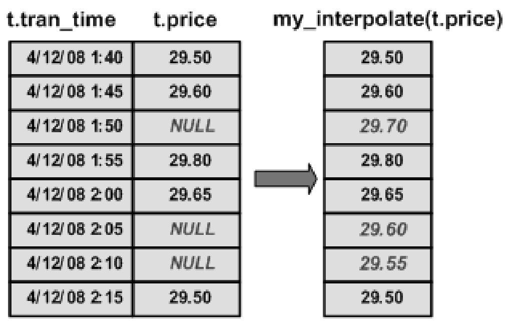
UDAF example: my_interpolate declaration

The “my_interpolate” example is an OLAP-style UDAF that attempts to fill in any missing values in a sequence (where missing values are denoted by NULLs) by performing linear interpolation across any set of adjacent NULL values to the nearest non-NULL value in each direction.

my_interpolate declaration

If the input at a given row is not NULL, the result for that row is the same as the input value.

Figure 2: my_interpolate results



t.tran_time	t.price
4/12/08 1:40	29.50
4/12/08 1:45	29.60
4/12/08 1:50	*NULL*
4/12/08 1:55	29.80
4/12/08 2:00	29.65
4/12/08 2:05	*NULL*
4/12/08 2:10	*NULL*
4/12/08 2:15	29.50

my_interpolate(t.price)
29.50
29.60
29.70
29.80
29.65
29.60
29.55
29.50

To operate at a sensible cost, my_interpolate must run using a fixed-width, row-based window, but the user can set the width of the window based on the maximum number of adjacent NULL values he or she expects to see. This function takes a set of double-precision floating point values and produces a resulting set of doubles.

The resulting UDAF declaration looks like this:

```
CREATE AGGREGATE FUNCTION my_interpolate (IN arg1 DOUBLE)
RETURNS DOUBLE
    OVER REQUIRED
    WINDOW FRAME REQUIRED
        RANGE NOT ALLOWED
        PRECEDING REQUIRED
        UNBOUNDED PRECEDING NOT ALLOWED
        FOLLOWING REQUIRED
        UNBOUNDED FOLLOWING NOT ALLOWED
    EXTERNAL NAME 'describe_my_interpolate@my_shared_lib'
```

OVER REQUIRED means that this function cannot be used as a simple aggregate (ON EMPTY INPUT, if used, is irrelevant).

WINDOW FRAME details specify that you must use a fixed-width, row-based window that extends both forward and backward from the current row when using this function. Because of these usage restrictions, my_interpolate is usable as an OLAP-style aggregate with an OVER clause similar to:

```
SELECT t.x,
    my_interpolate(t.x)
    OVER (ORDER BY t.x ROWS BETWEEN 5 PRECEDING AND 5 FOLLOWING)
        AS x_with_gaps_filled,
    COUNT(*)
FROM t
GROUP BY t.x
ORDER BY t.x
```

Within an OVER clause for my_interpolate, the precise number of preceding and following rows may vary, and optionally, you can use a PARTITION BY clause; otherwise, the rows must be similar to the example above given the usage restrictions in the declaration.

UDF Example: my_byte_length Declaration

my_byte_length is a simple scalar user-defined function that returns the size of a column in bytes.

my_byte_length declaration

When **my_byte_length** resides within the dynamically linkable library my_shared_lib, the declaration for this example is:

```
CREATE FUNCTION my_byte_length(IN arg1 LONG BINARY)
//             RETURNS UNSIGNED INT
//             DETERMINISTIC
//           IGNORE NULL VALUES
//             EXTERNAL NAME 'my_byte_length@libudfex'
```

This declaration says that **my_byte_length** is a simple scalar UDF residing in my_shared_lib with a descriptor routine named describe_my_byte_length. Since the behavior of a UDF may require more than one actual C/C++ entry point for its implementation, this set of entry points is not directly part of the **CREATE FUNCTION** syntax. Instead, the **CREATE FUNCTION** statement **EXTERNAL NAME** clause identifies a descriptor function for this UDF. A descriptor function, when invoked, returns a descriptor structure that is defined in detail in the next section. That descriptor structure contains the required and optional function pointers that embody the implementation of this UDF.

This declaration also says that **my_byte_length** accepts one `LONG BINARY` argument and returns an `UNSIGNED INT` result value.

Note: Large object data support requires a separately licensed Sybase IQ option.

The declaration states that this function is deterministic. A deterministic function always returns the identical result value when supplied the same input values. This means the result cannot depend on any external information beyond the supplied argument values, or on any side effects from previous invocations. By default, functions are assumed to be deterministic, so the results are the same if this characteristic is omitted from the **CREATE** statement.

The last piece of this declaration is the IGNORE NULL VALUES characteristic. Nearly all built-in scalar functions return a NULL result value if any of the input arguments are NULL. The IGNORE NULL VALUES states that the **my_byte_length** function follows that convention, and therefore this UDF routine is not actually invoked when either of its input values is NULL. Since RESPECT NULL VALUES is the default for functions, this characteristic must be specified in the declaration for this UDF to get the performance benefits. All functions that may return a non-NULL result given a NULL input value must use the default RESPECT NULL VALUES characteristic.

This example query with **my_byte_length** in the **SELECT** list returns a column with one row for each row in `exTable`, with an `INT` representing the size of the binary file:

```
SELECT my_byte_length(exLOBColumn)
FROM exTable
```

UDF Example: my_plus Declaration

The “my_plus” example is a simple scalar function that returns the result of adding its two integer argument values.

my_plus declaration

When my_plus resides within the dynamically linkable library my_shared_lib, the declaration for this example looks like this:

```
CREATE FUNCTION my_plus (IN arg1 INT, IN arg2 INT)
     RETURNS INT
     DETERMINISTIC
     IGNORE NULL VALUES
     EXTERNAL NAME 'my_plus@libudfex'
```

This declaration says that my_plus is a simple scalar UDF residing in my_shared_lib with a descriptor routine named describe_my_plus. Since the behavior of a UDF may require more than one actual C/C++ entry point for its implementation, this set of entry points is not directly part of the CREATE FUNCTION syntax. Instead, the CREATE FUNCTION statement EXTERNAL NAME clause identifies a descriptor function for this UDF. A descriptor function, when invoked, returns a descriptor structure that is defined in detail in the next section. That descriptor structure contains the required and optional function pointers that embody the implementation of this UDF.

This declaration says that my_plus accepts two INT arguments and returns an INT result value. If the function is invoked with an argument that is not an INT, and if the argument can be implicitly converted into an INT, the conversion happens before the function is called. If this function is invoked with an argument that cannot be implicitly converted into an INT, a conversion error is generated.

Further, the declaration states that this function is deterministic. A deterministic function always returns the identical result value when supplied the same input values. This means the result cannot depend on any external information beyond the supplied argument values, or on any side effects from previous invocations. By default, functions are assumed to be deterministic, so the results are the same if this characteristic is omitted from the CREATE statement.

The last piece of the above declaration is the IGNORE NULL VALUES characteristic. Nearly all built-in scalar functions return a NULL result value if any of the input arguments are NULL. The IGNORE NULL VALUES states that the my_plus function follows that convention, and therefore this UDF routine is not actually invoked when either of its input values are NULL. Since RESPECT NULL VALUES is the default for functions, this characteristic must be specified in the declaration for this UDF to get the performance benefits. All functions that may return a non-NULL result given a NULL input value must use the default RESPECT NULL VALUES characteristic.

In the following example query, my_plus appears in the SELECT list along with the equivalent arithmetic expression:

```
SELECT my_plus(t.x, t.y) AS x_plus_y_one, (t.x + t.y)AS x_plus_y_two
FROM t
WHERE t.z = 2
```

In the following example, my_plus is used in several different places and different ways within the same query:

```
SELECT my_plus(t.x, t.y), count(*)
FROM t
WHERE t.z = 2
AND my_plus(t.x, 5) > 10
AND my_plus(t.y, 5) > 10
GROUP BY my_plus(t.x, t.y)
```

UDF Example: my_plus_counter Declaration

The “my_plus_counter” example is a simple nondeterministic scalar UDF that takes a single integer argument, and returns the result of adding that argument value to an internal integer usage counter. If the input argument value is NULL, the result is the current value of the usage counter.

my_plus_counter declaration

Assuming that my_plus_counter also resides within the dynamically linkable library my_shared_lib, the declaration for this example is:

```
CREATE FUNCTION my_plus_counter (IN arg1 INT DEFAULT 0)
     RETURNS INT
     NOT DETERMINISTIC
     RESPECT NULL VALUES
     EXTERNAL NAME 'describe_my_plus_counter@my_shared_lib'
```

The RESPECT NULL VALUES characteristic means that this function is called even if the input argument value is NULL. This is necessary because the semantics of my_plus_counter includes:

- Internally keeping a usage count that increments even if the argument is NULL.
- A non-null value result when passed a NULL argument.

Because RESPECT NULL VALUES is the default, the results are the same if this clause is omitted from the declaration.

IQ restricts the usage of all nondeterministic functions. They are allowed only within the SELECT list of the top-level query block or in the SET clause of an UPDATE statement. They cannot be used within subqueries, or within a WHERE, ON, GROUP BY, or HAVING clause. This restriction applies to nondeterministic UDFs as well as to the nondeterministic built-in functions like GETUID and NUMBER.

The last detail in the above declaration is the DEFAULT qualifier on the input parameter. The qualifier tells the server that this function can be called with no arguments, and that when this happens the server automatically supplies a zero for the missing argument. If a DEFAULT value is specified, it must be implicitly convertible into the data type of that argument.

In the following example, the first SELECT list item adds the running counter to the value of t.x for each row. The second and third SELECT list items each return the same value for each row as the NUMBER function.

```
SELECT my_plus_counter(t.x),
     my_plus_counter(0),
     my_plus_counter(),
     NUMBER()
FROM t
```

TS_ARMA_AR Function [Time Series]

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested autoregressive estimate.

Syntax

TS_ARMA_AR (*timeseries_expression*, *ar_count*, *ar_elem*, *method*)

OVER (*window-spec*)

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_ARMA_CONST Function [Time Series]

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns an estimated constant.

Syntax

TS_ARMA_CONST (*timeseries_expression*, *method*)

OVER (*window-spec*)

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_ARMA_MA Function [Time Series]

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested moving average estimate.

Syntax

TS_ARMA_MA (*timeseries_expression*, *ma_count*, *ma_elem*, *method*)

OVER (*window-spec*)

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTOCORRELATION function [Time Series]

Calculates the sample autocorrelation function of a stationary time series.

Syntax

```
TS_AUTOCORRELATION (timeseries_expression, lagmax, lag_elem)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA Function [Time Series]

Determines parameters of a multiplicative seasonal autoregressive integrated moving average (ARIMA) model, and produces forecasts that incorporate the effects of outliers whose effects persist beyond the end of the series.

Syntax

```
TS_AUTO_ARIMA( <time_value>, <timeseries_expression>     [,<max_lag>[,
<critical    >    [,<epsilon>[,<criterion>    [,<confidence>[,<model>[,
<n_predictions>]]]]]]] )
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_OUTLIER Function [Time Series]

Like the **TS_AUTO_ARIMA** aggregate function, **TS_AUTO_ARIMA_OUTLIER** accepts an input time series and automatically determines the parameters of a multiplicative seasonal autoregressive integrated moving average (ARIMA) model.

Where **TS_AUTO_ARIMA** uses the ARIMA model to forecast values beyond the set of inputs, *TS_AUTO_ARIMA_OUTLIER* uses the ARIMA model to identify the elements of the input time series that are statistical outliers, and returns the outlier type of each one.

Syntax

```
TS_AUTO_ARIMA_OUTLIER( <time_value>, <timeseries_expression>     [,
<max_lag>[,<critical    >    [,<epsilon>[,<criterion>    [,<confidence>[,
<model>[,<delta>]]]]]]] )
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_AIC Function [Time Series]

Retrieves the Akaike's Information Criterion (AIC) output parameter produced by **TS_AUTO_ARIMA**.

Syntax

```
TS_AUTO_ARIMA_ RESULT_AIC(auto_arima_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_AICC Function [Time Series]

Retrieves the corrected AIC (AICC) output parameter produced by **TS_AUTO_ARIMA**.

Syntax

```
TS_AUTO_ARIMA_ RESULT_AICC(auto_arima_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_BIC Function [Time Series]

Retrieves the Bayesian Information Criterion (BIC) output parameter produced by **TS_AUTO_ARIMA**.

Syntax

```
TS_AUTO_ARIMA_ RESULT_BIC(auto_arima_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_FORECAST_VALUE Function [Time Series]

Retrieves the forecasted values for the requested outlier free series produced by **TS_AUTO_ARIMA**.

Syntax

```
TS_AUTO_ARIMA_RESULT_FORECAST_VALUE(auto_arima_result,
model_element_number)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_FORECAST_ERROR Function [Time Series]

Retrieves the forecasted standard error values for the original input series produced by **TS_AUTO_ARIMA**.

Syntax

TS_AUTO_ARIMA_RESULT_FORECAST_ERROR(*auto_arima_result***,** *forecast_element_number***)**

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_MODEL_D Function [Time Series]

Retrieves the *d* value produced by TS_AUTO_ARIMA when computing the ARIMA model description.

Syntax

TS_AUTO_ARIMA_ RESULT_MODEL_D(*auto_arima_result***)**

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_MODEL_P Function [Time Series]

Retrieves the *p* value produced by TS_AUTO_ARIMA when computing the ARIMA model description.

Syntax

TS_AUTO_ARIMA_ RESULT_MODEL_P(*auto_arima_result***)**

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_MODEL_Q [Time Series]

Retrieves the *q* value produced by TS_AUTO_ARIMA when computing the ARIMA model description.

Syntax

TS_AUTO_ARIMA_ RESULT_MODEL_Q(*auto_arima_result***)**

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_MODEL_S function [Time Series]

Retrieves the *s* value produced by TS_AUTO_ARIMA when computing the ARIMA model description.

Syntax

```
TS_AUTO_ARIMA_ RESULT_MODEL_S(auto_arima_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_ARIMA_RESULT_RESIDUAL_SIGMA [Time Series]

Retrieves the residual standard error of the outlier-free data points.

Syntax

```
TS_AUTO_ARIMA_ RESULT_RESIDUAL_SIGMA (auto_arima_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_AUTO_UNI_AR Function [Time Series]

Performs automatic selection and fitting of a univariate autoregressive time series model.

Syntax

```
TS_AUTO_UNI_AR (timeseries_expression, ar_count, ar_elem, method)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_BOX_COX_XFORM Function [Time Series]

Performs a forward or inverse Box-Cox power transformation.

Syntax

```
TS_BOX_COX_XFORM (timeseries_expression, power [, shift [,
inverse] ]) OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_DIFFERENCE Function [Time Series]

Differences a seasonal or nonseasonal time series.

Syntax

```
TS_DIFFERENCE (timeseries_expression, period1 [, period2 [, ...period
10] ]) OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_DOUBLE_ARRAY [Time Series]

A supporting function for the TS_GARCH function. Constructs a logical array consisting of three to ten constant double precision floating point values, and returns a single varbinary value.

Syntax

```
TS_DOUBLE_ARRAY(xguess1, xguess2, xguess3, [ … [ , xguess10] …] ])
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_ESTIMATE_MISSING Function [Time Series]

Estimates the missing values in a time series and returns them as a new time series, interspersed with the original time series.

Syntax

```
TS_ESTIMATE_MISSING (timeseries_expression, method)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_GARCH Function [Time Series]

Computes estimates of the parameters of a GARCH (p, q) model.

Syntax

```
TS_GARCH (<time series expression>, <garch_count>, <arch_count>,
<xguess_binary_encoding>, [, <max_sigma>])
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_GARCH_RESULT_A Function [Time Series]

A supporting function for the TS_GARCH function. Retrieves the Log-Likelihood output parameter, *A*, produced by the **TS_GARCH** aggregate function.

Syntax

```
TS_GARCH_RESULT_A (ts_garch_result )
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_GARCH_RESULT_AIC Function [Time Series]

A supporting function for the TS_GARCH function. Retrieves the Akaike Information Criterion output parameter, *AIC*, produced by the **TS_GARCH** aggregate function.

Syntax

```
TS_GARCH_RESULT_AIC (ts_garch_result)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the Time Series Guide for detailed information on this function.

TS_GARCH_RESULT_USER [Time Series]

A supporting function for the **TS_GARCH** function. Accesses each element in the logical array that describes the `GARCH(p,q)` model.

Syntax

```
TS_GARCH_RESULT_USER (ts_garch_result,model_element_number)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_INT_ARRAY [Time Series]

A supporting function for the TS_AUTO_ARIMA function and the TS_AUTO_ARIMA_OUTLIER function. Constructs a logical array of constant integer values encoded as a varbinary value.

Syntax

```
TS_INT_ARRAY(int1,int2,int3,int4,[ … [ ,  int10] …] ])
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_LACK_OF_FIT Function [Time Series]

Performs the lack-of-fit test for a univariate time series or transfer function, given the appropriate correlation function.

Syntax

```
TS_LACK_OF_FIT (timeseries_expression, p_value, q_value, lagmax,
[tolerance])
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_LACK_OF_FIT_P Function [Time Series]

Performs the lack-of-fit test for a univariate time series. This function is identical to the TS_LACK_OF_FIT function, except that it returns the p-value of q, rather than returning q.

Syntax

```
TS_LACK_OF_FIT_P (timeseries_expression, p_value, q_value, lagmax,
[tolerance])
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_MAX_ARMA_AR Function [Time Series]

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested autoregressive estimate.

Syntax

```
TS_MAX_ARMA_AR (timeseries_expression, ar_count, ar_elem)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_MAX_ARMA_CONST Function [Time Series]

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the constant estimate.

Syntax

```
TS_MAX_ARMA_CONST (timeseries_expression)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_MAX_ARMA_LIKELIHOOD Function [Time Series]

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns likelihood value (ln) for the fitted model.

Syntax

```
TS_MAX_ARMA_LIKELIHOOD (timeseries_expression)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_MAX_ARMA_MA Function [Time Series]

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested moving average estimate.

Syntax

```
TS_MAX_ARMA_MA (timeseries_expression, ma_count, ma_elem)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_OUTLIER_IDENTIFICATION Function [Time Series]

Detects and determines outliers and simultaneously estimates the model parameters in a time series where the underlying outlier-free series follows a general seasonal or non-seasonal ARMA model.

Syntax

```
TS_OUTLIER_IDENTIFICATION (timeseries_expression, p_value, q_value,
s_value, d_value, [, delta_value[, critical_value]])
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

TS_PARTIAL_AUTOCORRELATION Function [Time Series]

Calculates the sample partial autocorrelation function of a stationary time series.

Syntax

```
TS_PARTIAL_AUTOCORRELATION (timeseries_expression, lagmax, lag_elem)
```

```
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

ARGN Function [Miscellaneous]

Returns a selected argument from a list of arguments.

Syntax

```
ARGN ( integer-expression, expression [ , …] )
```

Parameters

Table 71. Parameters

Parameter	Description
integer-expression	The position of an argument within a list of ex-pressions.
expression	n expression of any data type passed into the function. All supplied expressions must be of the same data type.

Returns

Using the value of the *integer-expression* as *n*, returns the nth argument (starting at 1) from the remaining list of arguments.

Example

The following statement returns the value 6:

```
SELECT ARGN( 6, 1,2,3,4,5,6 ) FROM iq_dummy
```

Usage

Using the value of *integer-expression* as *n* returns the *n*th argument (starting at 1) from the remaining list of arguments. While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

COALESCE Function [Miscellaneous]

Returns the first non-NULL expression from a list.

Syntax

```
COALESCE ( expression, expression [ , … ] )
```

Parameters

Table 72. Parameters

Parameter	Description
expression	Any expression.

Returns

ANY

Example

The following statement returns the value 34:

```
SELECT COALESCE( NULL, 34, 13, 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

ERRORMSG Function [Miscellaneous]

Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

Syntax

```
ERRORMSG ( [ sqlstate | sqlcode ] )
```

```
sqlstate: string
```

```
sqlcode: integer
```

Parameters

Table 73. Parameters

Parameter	Definition
sqlstate	The SQLSTATE value for which the error message is to be returned.
sqlcode	The SQLCODE value for which the error message is to be returned.

Returns

A string containing the error message.

VARCHAR

Example

The following statement returns the error message for SQLCODE -813:

```
select errormsg( -813 )
```

Return Value

A string containing the error message. If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.

If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders ('???').

The **ERRORMSG** function returns SQL Anywhere and Sybase IQ error messages.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

IFNULL Function [Miscellaneous]

Returns the first nonnull expression, or NULL.

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, then the NULL value is returned.

Syntax

```
IFNULL ( expression1, expression2 [ , expression3 ] )
```

Parameters

Table 74. Parameters

Parameter	Description
expression1	The expression to be evaluated. Its value determines whether *expression2* or *expression3* is returned.
expression2	The return value if *expression1* is NULL
expression3	The return value if *expression1* is not NULL.

Returns

The data type returned depends on the data type of *expression-2* and *expression-3*.

Examples

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 ) FROM iq_dummy
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 ) FROM iq_dummy
```

Standards and compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

ISNULL Function [Miscellaneous]

Returns the value of the first non-NULL expression in the parameter list.

At least two expressions must be passed to the function.

Syntax

```
ISNULL ( expression, expression [ …, expression ] )
```

Parameters

Table 75. Parameters

Parameter	Description
expression	An expression to be tested against NULL.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

Example

The following statement returns the value -66:

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 ) FROM iq_dummy
```

Usage

The **ISNULL** function is the same as the **COALESCE** function.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

ISNUMERIC Function [Miscellaneous]

Tests whether a string argument can be converted to a numeric.

If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax

```
ISNUMERIC ( string )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 76. Parameters

Parameter	Description
string	The string to be analyzed to determine whether the string represents a valid numeric value

Returns

INT

Usage

For optimal performance, avoid using **ISNUMERIC** in predicates, where it is processed by the SQL Anywhere portion of the product and cannot take advantage of the performance features of Sybase IQ.

Example

The following example tests whether the `height_in_cms` column holds valid numeric data, returning invalid numeric data as NULL, and valid numeric data in `int` format.

```
data height_in_cms
------------------------
asde
asde
180
156
```

```
select case
   when isnumeric(height_in_cms)=0
   then NULL
   else cast(height_in_cms as int)
   end
from MyData
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

NEWID Function [Miscellaneous]

Generates a UUID (Universally Unique Identifier) value.

The returned UUID value is a binary. A UUID is the same as a GUID (Globally Unique Identifier).

Syntax

```
NEWID ( )
```

Parameters

There are no parameters associated with **NEWID**().

Returns

UNIQUEIDENTIFIER

Example

The following statement creates the table `t1` and then updates the table, setting the value of the column `uid_col` to a unique identifier generated by the **NEWID** function, if the current value of the column is NULL.

```
CREATE TABLE t1 (uid_col int);
UPDATE t1
    SET uid_col = NEWID()
      WHERE uid_col IS NULL
```

If you execute the following statement,

```
SELECT NEWID()
```

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

Usage

The **NEWID**() function generates a unique identifier value.

UUIDs can be used to uniquely identify objects in a database. The values are generated such that a value produced on one computer does not match that produced on another, hence they can also be used as keys in replication and synchronization environments.

The **NEWID** function is supported only in the following positions:

- **SELECT** list of a top level query block
- **SET** clause of an **UPDATE** statement
- **VALUES** clause of **INSERT...VALUES**

You can use a value generated by the **NEWID** function as a column default value in a Sybase IQ table.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

NULLIF Function [Miscellaneous]

Provides an abbreviated **CASE** expression by comparing expressions.

Syntax

```
NULLIF ( expression1, expression2 )
```

Parameters

Parameter	Description
expression1	An expression to be compared.
expression2	An expression to be compared.

Returns

Data type of the first argument.

Examples

The following statement returns a:

```
SELECT NULLIF( 'a', 'b' ) FROM iq_dummy
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' ) FROM iq_dummy
```

Usage

NULLIF compares the values of the two expressions.

If the first expression equals the second expression, **NULLIF** returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, **NULLIF** returns the first expression.

The **NULLIF** function provides a short way to write some **CASE** expressions. **NULLIF** is equivalent to:

```
CASE WHEN expression1 = expression2 THEN NULL
ELSE expression1 END
```

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

ROWID Function [Miscellaneous]

Returns the internal row ID value for each row of the table.

Syntax

```
ROWID ( table-name ) …FROM table-name
```

Parameters

Parameter	Description
table-name	The name of the table. Specify the name of the table within the parentheses with either no quotes or with double quotes.

Returns

UNSIGNED BIGINT

Examples

The following statement returns the row ID values 1 through 10:

```
SELECT ROWID( “PRODUCTS” ) FROM PRODUCTS
```

rowid(Products)
1
2
3
.
.
.
10

The following statement returns the product ID and row ID value of all rows with a product ID value less than 400:

```
SELECT PRODUCTS.ID, ROWID ( PRODUCTS )
FROM PRODUCTS
WHERE PRODUCTS.ID < 400
```

ID	rowid(Products)
300	1
301	2
302	3

The following statement deletes all rows with row ID values greater than 50:

```
DELETE FROM PRODUCTS
WHERE ROWID ( PRODUCTS ) > 50
```

Usage

You can use the **ROWID** function with other clauses to manipulate specific rows of the table.

You must specify the **FROM** *table-name* clause.

A limitation of the **ROWID** function is that it cannot use a join index of that table, eliminating any performance benefits that would normally use that join index.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SQLFLAGGER Function [Miscellaneous]

Returns the conformity of a given SQL statement to a specified standard.

Syntax

```
SQLFLAGGER ( sql-standard-string, sql-statement-string )
```

Parameters

Parameter	Description
sql-standard-string	The standard level against which to test compliance. Possible values are the same as for the SQL_FLAGGER_ERROR_LEVEL database option: • SQL:2003/Core Test for conformance to core SQL/2003 syntax. • SQL:2003/Package Test for conformance to full SQL/2003 syntax. • SQL:1999/Core Test for conformance to core SQL/1999 syntax. • SQL:1999/Package Test for conformance to full SQL/1999 syntax. • SQL:1992/Entry Test for conformance to entry-level SQL/1992 syntax. • SQL:1992/Intermediate Test for conformance to intermediate-level SQL/1992 syntax. • SQL:1992/Full Test for conformance to full-SQL/1992 syntax.
sql-statement-string	The SQL statement to check for conformance.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SQLFLAGGER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SQLFLAGGER** to the correct data type and size.

Examples

The following statement shows an example of the message that is returned when a disallowed extension is found:

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT top 1 dummy_col FROM
sys.dummy ORDER BY dummy_col' );
```

This statement returns the message `'0AW03 Disallowed language extension detected in syntax near 'top' on line 1'`.

The following statement returns '00000' because it contains no disallowed extensions:

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT dummy_col FROM
sys.dummy' );
```

Usage

You can also use the iqsqlpp SQL Preprocessor Utility to flag any Embedded SQL that is not part of a specified set of SQL92. See *Utility Guide > iqsqlpp SQL Preprocessor Utility*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

COL_NAME Function [System]

Returns the column name.

Syntax

```
COL_NAME ( table-id, column-id [ , database-id ] )
```

Parameters

Table 77. Parameters

Parameter	Description
table-id	The object ID of the table.
column-id	The column ID of the column.
database-id	The database ID.

Examples

The following statement returns the column name `lname`. The object ID of the `Customers` table is 100209, as returned by the **OBJECT_ID** function. The column ID is stored in the `column_id` column of the `syscolumn` system table. The database ID of the `iqdemo` database is 0, as returned by the DB_ID function.

```
SELECT COL_NAME( 100209, 3, 0 ) FROM iq_dummy
```

The following statement returns the column name **city**.

```
SELECT COL_NAME ( 100209, 5 )FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

TS_VWAP Function [Time Series]

VWAP stands for volume-weighted average price. **TS_VWAP** calculates the ratio of the value traded to the total volume traded over a particular time horizon.

VWAP is a measure of the average price of a stock over a defined trading horizon. You can use TS_VWAP as both a simple and an OLAP-style aggregate function.

Unlike the other time series functions, TS_VWAP does not call the IMSL libraries.

Syntax 1

```
TS_VWAP (price_expression, volume_expression)
```

Syntax 2

```
TS_VWAP (price_expression, volume_expression)
OVER (window-spec)
```

Note: This function is available only with RAP – The Trading Edition Enterprise. See the *Time Series Guide* for detailed information on this function.

CONNECTION_PROPERTY Function [System]

Returns the value of a given connection property as a string.

Syntax

```
CONNECTION_PROPERTY ( { integer-expression1 | string-expression }
                                                      … [ , integer-
expression2 ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 78. Parameters

Parameter	Description
integer-expression1	In most cases, it is more convenient to supply a string expression as the first argument. If you do supply integer-expression1, it is the connection property ID. You can determine this using the PROPERTY_NUMBER function.
string-expression	The connection property name. You must specify either the property ID or the property name.

Parameter	Description
integer-expression2	The connection ID of the current database connection. The current connection is used if this argument is omitted.

Returns

VARCHAR

Example

The following statement returns the number of prepared statements being maintained, for example, 4:

```
SELECT connection_property( 'PrepStmt' )FROM iq_dummy
```

Usage

The current connection is used if the second argument is omitted.

Standards and Compatibility

- Vendor extension to ISO/ANSI SQL grammar.
- Compatible with Adaptive Server Enterprise

DATALENGTH Function [System]

Returns the length of the expression in bytes.

Syntax

DATALENGTH (*expression*)

Parameters

Parameter	Description
expression	The expression is usually a column name. If the expression is a string constant, it must be enclosed in quotes.

Returns

UNSIGNED INT

Usage

Table 79. DATALENGTH return values

Data type	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	Length of the data
BINARY	Length of the data

Example

Returns the value 35, the longest string in the `company_name` column:

```
SELECT MAX( DATALENGTH( company_name ) )
FROM Customers
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

DB_ID Function [System]

Returns the database ID number.

Syntax

DB_ID ([*database-name*])

Note: CIS functional compensation performance considerations apply.

Parameters

Table 80. Parameters

Parameter	Description
database-name	A string expression containing the database name. If database-name is a string constant, it must be enclosed in quotes. If no database-name is supplied, the ID number of the current database is returned

Returns

INT

Examples

Returns the value 0, if `iqdemo` is the only running database:

```
SELECT DB_ID( 'iqdemo' ) FROM iq_dummy
```

Returns the value 0, if executed against the only running database:

```
SELECT DB_ID() FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

DB_NAME Function [System]

Returns the database name.

Syntax

```
DB_NAME ( [ database-id ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 81. Parameters

Parameter	Description
database-id	The ID of the database. database-id must be a numeric expression

Returns

VARCHAR

Example

Returns the database name **iqdemo**, when executed against the demo database:

```
SELECT DB_NAME( 0 ) FROM iq_dummy
```

Usage

If no *database-id* is supplied, the name of the current database is returned.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

EVENT_CONDITION_NAME Function [System]

Can be used to list the possible parameters for **EVENT_CONDITION**.

To define an event and its associated handler, use the **CREATE EVENT** statement.

See *Reference: Statements and Options > SQL Statements > CREATE EVENT Statement*.

Syntax

```
EVENT_CONDITION_NAME ( integer )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
integer	Must be greater than or equal to zero.

Returns

VARCHAR

Usage

You can use **EVENT_CONDITION_NAME** to obtain a list of all **EVENT_CONDITION** arguments by looping over integers until the function returns NULL.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

EVENT_PARAMETER Function [System]

Provides context information for event handlers.

To define an event and its associated handler, use the **CREATE EVENT** statement.

See *Reference: Statements and Options > SQL Statements > CREATE EVENT statement*.

Syntax

```
EVENT_PARAMETER ( context-name )
```

```
context-name:
  'ConnectionID'
```

```
| 'User'
| 'EventName'
| 'Executions'
| 'IQDBMainSpaceName'
| 'NumActive'
| 'TableName'
| condition-name
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 82. Parameters

Parameter	Description
context-name	One of the preset strings. The strings are case-insensitive, and carry the following information:
ConnectionId	The connection ID, as returned by `connection_property( 'id' )`
User	The user ID for the user that caused the event to be triggered.
EventName	The name of the event that has been triggered.
Executions	The number of times the event handler has been executed.
NumActive	The number of active instances of an event handler. This is useful if you want to limit an event handler so that only one instance executes at any given time.
TableName	The name of the table, for use with RemainingValues.

In addition, you can access any of the valid *condition-name* arguments to the **EVENT_CONDITION** function from the **EVENT_PARAMETER** function.

Returns

VARCHAR

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

GROUP_MEMBER Function [System]

Identifies whether the user belongs to the specified group.

Syntax

```
GROUP_MEMBER ( group-name-string-expression[ , user-name-string-
expression ] )
```

Parameters

Parameter	Description
group-name-string-expression	Identifies the group to be considered.
user-name-string-expression	Identifies the user to be considered. If not supplied, then the current user name is assumed.

Return Values

Table 83. Return Values

Value	Description
0	Returns 0 if the group does not exist, if the user does not exist, or if the user does not belong to the specified group.
1	Returns an integer other than 0 if the user is a member of the specified group.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

INDEX_COL Function [System]

Returns the name of the indexed column.

Syntax

```
INDEX_COL ( table-name, index-id, key_# [ , user-id ] )
```

Parameters

Parameter	Definition
table-name	A table name.

Parameter	Definition
index-id	The index ID of an index of *table-name*.
key_#	A key in the index specified by *index-id*. This parameter specifies the column number in the index. For a single column index, *key_#* is equal to 0. For a multicolumn index, *key_#* is equal to 0 for the first column, 1 for the second column, and so on.
user-id	The user ID of the owner of *table-name*. If *user-id* is not specified, this value defaults to the caller's user ID..

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

NEXT_CONNECTION Function [System]

Returns the next connection number, or the first connection if the parameter is NULL.

Syntax

```
NEXT_CONNECTION ( {connection-id }, {database-id } )
```

Note: CIS functional compensation performance considerations apply.

Returns

INT

Parameters

Parameter	Description
connection-id	An integer, usually returned from a previous call to **NEXT_CONNECTION**. If *connection-id* is NULL, **NEXT_CONNECTION** returns the most recent connection ID.
database-id	An integer representing one of the databases on the current server. If you supply no *database-id*, the current database is used. If you supply NULL, then **NEXT_CONNECTION** returns the next connection regardless of database.

Usage

You can use **NEXT_CONNECTION** to enumerate the connections to a database. To get the first connection, pass NULL; to get each subsequent connection, pass the previous return value. The function returns NULL when there are no more connections.

NEXT_CONNECTION can be used to enumerate the connections to a database. Connection IDs are generally created in monotonically increasing order. This function returns the next connection ID in reverse order.

To get the connection ID value for the most recent connection, enter NULL as the *connection-id*. To get the subsequent connection, enter the previous return value. The function returns NULL when there are no more connections in the order.

NEXT_CONNECTION is useful if you want to disconnect all the connections created before a specific time. However, because **NEXT_CONNECTION** returns the connection IDS in reverse order, connections made after the function is started are not returned. If you want to ensure that all connections are disconnected, prevent new connections from being created before you run **NEXT_CONNECTION**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

Examples

The following statement returns an identifier for the first connection on the current database. The identifier is an integer value like 10.

```
SELECT NEXT_CONNECTION( NULL );
```

The following statement returns a value like 5.

```
SELECT NEXT_CONNECTION( 10 );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* on the current database.

```
SELECT NEXT_CONNECTION( connection-id );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* (regardless of database).

```
SELECT NEXT_CONNECTION( connection-id, NULL );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* on the specified database.

```
SELECT NEXT_CONNECTION( connection-id, database-id );
```

The following call returns the first (earliest) connection (regardless of database).

```
SELECT NEXT_CONNECTION( NULL, NULL );
```

The following call returns the first (earliest) connection on the specified database.

```
SELECT NEXT_CONNECTION( NULL, database-id );
```

NEXT_DATABASE Function [System]

Returns the next database ID number, or the first database if the parameter is NULL.

Syntax

```
NEXT_DATABASE ( { NULL | database-id } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
database-id	An integer that specifies the ID number of the database.

Returns

INT

Examples

The following statement returns the value 0, the first database value:

```
SELECT NEXT_DATABASE( NULL ) FROM iq_dummy
```

The following statement returns NULL, indicating that there are no more databases on the server:

```
SELECT NEXT_DATABASE( 0 ) FROM iq_dummy
```

Usage

You can use **NEXT_DATABASE** to enumerate the databases running on a database server. To get the first database, pass NULL; to get each subsequent database, pass the previous return value. The function returns NULL when there are no more databases.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

OBJECT_ID Function [System]

Returns the object ID.

Syntax

```
OBJECT_ID ( object-name )
```

Parameters

Table 84. Parameters

Parameter	Description
object-name	The name of the object.

Examples

The following statement returns the object ID 100209 of the *Customers* table:

```
SELECT OBJECT_ID ('CUSTOMERS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

OBJECT_NAME Function [System]

Returns the object name.

Syntax

OBJECT_NAME (*object-id* [, *database-id*])

Parameters

Table 85. Parameters

Parameter	Description
object-id	The object ID.
database-id	The database ID.

Examples

The following statement returns the name “customer:”

```
SELECT OBJECT_NAME ( 100209 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

PROPERTY Function [System]

Returns the value of the specified server-level property as a string.

Syntax

```
PROPERTY ( { property-id  |  property-name } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 86.

Parameter	Description
property-id	An integer that is the property-number of the server-level property. This number can be determined from the **PROPERTY_NUMBER** function. The *property-id* is commonly used when looping through a set of properties.
property-name	A string giving the name of the property.

Returns

VARCHAR

Example

The following statement returns the name of the current database server:

```
SELECT PROPERTY( 'Name' ) FROM iq_dummy
```

Usage

Each property has both a number and a name, but the number is subject to change between versions, and should not be used as a reliable identifier for a given property.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

PROPERTY_DESCRIPTION Function [System]

Returns a description of a property.

Syntax

```
PROPERTY_DESCRIPTION ( { property-id | property-name } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-id	An integer that is the property number of the property. This number can be determined from the **PROPERTY_NUMBER** function. The *property-id* is commonly used when looping through a set of properties.
property-name	A string giving the name of the property.

Returns

VARCHAR

Example

The following statement returns the description "Number of index insertions:"

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' ) FROM iq_dummy
```

Usage

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

PROPERTY_NAME Function [System]

Returns the name of the property with the supplied property number.

Syntax

```
PROPERTY_NAME ( property-id )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-id	The property number of the property.

Returns

VARCHAR

Example

The following statement returns the property associated with property number 126. The actual property to which this refers changes from version to version.

```
SELECT PROPERTY_NAME( 126 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

PROPERTY_NUMBER Function [System]

Returns the property number of the property with the supplied property name.

Syntax

```
PROPERTY_NUMBER ( property-name )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-name	A property name.

Returns

INT

Example

The following statement returns an integer value. The actual value changes from version to version.

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SUSER_ID Function [System]

Returns an integer user identification number.

Syntax

```
SUSER_ID ( [ user-name ] )
```

Parameters

Parameter	Description
user-name	The user name.

Returns

INT

Examples

The following statement returns the user identification number 1:

```
SELECT SUSER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT SUSER_ID ('SYS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

SUSER_NAME Function [System]

Returns the user name.

Syntax

```
SUSER_NAME ( [ user-id ] )
```

Parameters

Parameter	Description
user-id	The user identification number.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SUSER_NAME** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SUSER_NAME** to the correct data type and size.

Examples

The following statement returns the value DBA:

```
SELECT SUSER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value SYS:

```
SELECT SUSER_NAME ( 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, **SUSER_NAME** returns the server user name.

USER_ID Function [System]

Returns an integer user identification number.

Syntax

```
USER_ID ( [ user-name ] )
```

Parameters

Parameter	Description
user-name	The user name.

Returns

INT

Examples

The following statement returns the user identification number 1:

```
SELECT USER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT USER_ID ('SYS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

USER_NAME Function [System]

Returns the user name.

Syntax

```
USER_NAME ( [ user-id ] )
```

Parameters

Parameter	Description
user-id	The user identification number.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **USER_NAME** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **USER_NAME** to the correct data type and size.

Examples

The following statement returns the value “DBA”:

```
SELECT USER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value “SYS”:

```
SELECT USER_NAME ( 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, USER_NAME returns the user name, not the server user name.

COL_NAME Function [System]

Returns the column name.

Syntax

```
COL_NAME ( table-id, column-id [ , database-id ] )
```

Parameters

Table 87. Parameters

Parameter	Description
table-id	The object ID of the table.
column-id	The column ID of the column.
database-id	The database ID.

Examples

The following statement returns the column name `lname`. The object ID of the `Customers` table is 100209, as returned by the **OBJECT_ID** function. The column ID is stored in the `column_id` column of the `syscolumn` system table. The database ID of the `iqdemo` database is 0, as returned by the DB_ID function.

```
SELECT COL_NAME( 100209, 3, 0 ) FROM iq_dummy
```

The following statement returns the column name **city**.

```
SELECT COL_NAME ( 100209, 5 )FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for Sybase IQ.

HEXTOBIGINT Function [Data Type Conversion]

Returns the `BIGINT` equivalent of a hexadecimal string.

Syntax

```
HEXTOBIGINT ( hexadecimal-string )
```

Parameters

Parameter	Description

hexadecimal-string	The hexadecimal value to be converted to a big integer (`BIGINT`). Input can be in the following forms, with either a lowercase or uppercase “0x” in the prefix, or no prefix:
	`0x`*`hex-string`*
	`0X`*`hex-string`*
	`hex-string`

Examples

The following statements return the value 4294967287:

```
SELECT HEXTOBIGINT ( '0xfffffff7' ) FROM iq_dummy
```

```
SELECT HEXTOBIGINT ( '0Xfffffff7' ) FROM iq_dummy
```

```
SELECT HEXTOBIGINT ( 'fffffff7' ) FROM iq_dummy
```

Usage

The **HEXTOBIGINT** function accepts hexadecimal integers and returns the `BIGINT` equivalent. Hexadecimal integers can be provided as `CHAR` and `VARCHAR` value expressions, as well as `BINARY` and `VARBINARY` expressions.

The **HEXTOBIGINT** function accepts a valid hexadecimal string, with or without a “0x” or “0X” prefix, enclosed in single quotes.

Input of fewer than 16 digits is assumed to be left-padded with zeros.

For data type conversion failure on input, Sybase IQ returns an error unless the `CONVERSION_ERROR` option is set to OFF. When `CONVERSION_ERROR` is OFF, invalid hexadecimal input returns NULL.

See Reference: Statements and Options > Database Options > Alphabetical List of Options > CONVERSION_ERROR Option [TSQL].

An error is returned if a `BINARY` or `VARBINARY` value exceeds 8 bytes and a `CHAR` or `VARCHAR` value exceeds 16 characters, with the exception of the value being appended with ‘0x.’

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

HEXTOINT Function [Data Type Conversion]

Returns the unsigned `BIGINT` equivalent of a hexadecimal string.

Syntax

```
HEXTOINT ( hexadecimal-string )
```

Parameters

Parameters	Description

hexadecimal-string	The string to be converted to an integer. Input can be in the following forms, with either a lowercase or uppercase "x" in the prefix, or no prefix:
	`0xhex-string`
	`0Xhex-string`
	`hex-string`

Returns

The HEXTOINT function returns the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8-9 and the uppercase or lowercase letters A-F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT( '0x0080000001' );
```

INT

Examples

The following statements return the value 420:

```
SELECT HEXTOINT ( '0x1A4' ) FROM iq_dummy
```

```
SELECT HEXTOINT ( '0X1A4' ) FROM iq_dummy
```

```
SELECT HEXTOINT ( '1A4' ) FROM iq_dummy
```

Usage

For invalid hexadecimal input, Sybase IQ returns an error unless the `CONVERSION_ERROR` option is OFF. When `CONVERSION_ERROR` is OFF, invalid hexadecimal input returns NULL.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > CONVERSION_ERROR Option [TSQL]*.

The database option `ASE_FUNCTION_BEHAVIOR` specifies that output of Sybase IQ functions, including **INTTOHEX** and **HEXTOINT**, is consistent with the output of Adaptive Server Enterprise functions.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > ASE_FUNCTION_BEHAVIOR Option*.

When the `ASE_FUNCTION_BEHAVIOR` option is ON:

- Sybase IQ **HEXTOINT** assumes input is a hexadecimal string of 8 characters; if the length is less than 8 characters long, the string is left padded with zeros.
- Sybase IQ **HEXTOINT** accepts a maximum of 16 characters prefixed with 0x (a total of 18 characters); use caution, as a large input value can result in an integer value that overflows the 32-bit signed integer output size.
- The data type of the output of the Sybase IQ **HEXTOINT** function is assumed to be a 32-bit signed integer.
- Sybase IQ **HEXTOINT** accepts a 32-bit hexadecimal integer as a signed representation.
- For more than 8 hexadecimal characters, Sybase IQ **HEXTOINT** considers only relevant characters.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

INTTOHEX Function [Data Type Conversion]

Returns the hexadecimal equivalent of a decimal integer.

Syntax

```
INTTOHEX ( integer-expression )
```

Parameters

Table 88. Parameters

Parameter	Description
integer-expression	The integer to be converted to hexadecimal.

Returns

VARCHAR

Examples

The following statement returns the value 3B9ACA00:

```
SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
```

The following statement returns the value 00000002540BE400:

```
SELECT INTTOHEX ( 10000000000) FROM iq_dummy
```

Usage

If data conversion of input to **INTTOHEX** conversion fails, Sybase IQ returns an error, unless the `CONVERSION_ERROR` option is OFF. In that case, the result is NULL.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > CONVERSION_ERROR Option [TSQL].*

The database option `ASE_FUNCTION_BEHAVIOR` specifies that output of Sybase IQ functions, including **INTTOHEX** and **HEXTOINT**, be consistent with the output of Adaptive Server Enterprise functions. The default value of `ASE_FUNCTION_BEHAVIOR` is OFF.

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > ASE_FUNCTION_BEHAVIOR Option*.

When the `ASE_FUNCTION_BEHAVIOR` option is disabled (the value is OFF):

- The output of **INTTOHEX** is compatible with SQL Anywhere.
- Depending on the input, the output of **INTTOHEX** can be 8 digits or 16 digits and is left padded with zeros; the return data type is `VARCHAR`.
- The output of **INTTOHEX** does not have a ‘0x’ or ‘0X’ prefix.
- The input to **INTTOHEX** can be up to a 64-bit integer.

When the `ASE_FUNCTION_BEHAVIOR` option is enabled (the value is ON):

- The output of **INTTOHEX** is compatible with ASE.
- The output of **INTTOHEX** is always 8 digits and is left-padded with zeros; the return data type is `VARCHAR.`
- The output of **INTTOHEX** does not have a ‘0x’ or ‘0X’ prefix.
- Sybase IQ **INTTOHEX** assumes input is a 32-bit signed integer; a larger value can overflow and a conversion error can result. For example, the statement:

  ```
  SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
  ```

  returns the value `3B9ACA00`. But the statement:

  ```
  SELECT INTTOHEX( 10000000000 ) FROM iq_dummy
  ```

  results in a conversion error.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

CUME_DIST Function [Analytical]

The **CUME_DIST** function is a rank analytical function that calculates the relative position of one value among a group of rows. It returns a decimal value between 0 and 1.

Syntax

```
CUME_DIST () OVER (window-spec)
```

Returns

A DOUBLE value between 0 and 1

Example

The following example returns a result set that provides a cumulative distribution of the salaries of employees who live in California:

```
SELECT DepartmentID, Surname, Salary,CUME_DIST() OVER (PARTITION BY
DepartmentIDORDER BY Salary DESC) "Rank"FROM Employees WHERE State IN
('CA');
```

The returned result set is:

Table 89. CUME_DIST result set

DepartmentID	Surname	Salary	Rank
200	Savarino	72,300.000	0.333333
200	Clark	45,000.000	0.666667
200	Overbey	39,300.000	1.000000

Usage

Sybase IQ calculates the cumulative distribution of a value of x in a set S of size N using:CUME_DIST(x) = number of values in S coming before and including x in the specified order / N

Composite sort-keys are not currently allowed in the **CUME_DIST** function. You can use composite sort-keys with any of the other rank functions.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement. The *window-spec* must contain the **ORDER BY** clause, and cannot contain a **ROWS** or **RANGE** clause.

Note: DISTINCT is not supported.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL feature T612.

- Sybase—Compatible with SQL Anywhere.

DENSE_RANK Function [Analytical]

Ranks items in a group.

Syntax

DENSE_RANK ( ) **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

Table 90. Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

INTEGER

Example

The following statement illustrates the use of the **DENSE_RANK** function:

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       rank_dense
supplier#011     200,000            1
supplier#002     200,000            1
supplier#013     123,000            2
supplier#004     110,000            3
supplier#035     110,000            3
supplier#006     50,000             4
supplier#021     10,000             5
```

Usage

DENSE_RANK is a rank analytical function. The dense rank of row R is defined as the number of rows preceding and including R that are distinct within the groups specified in the **OVER** clause or distinct over the entire result set. The difference between **DENSE_RANK** and **RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

DENSE_RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is *not* an **ORDER BY**

for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

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

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

DENSE_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **DENSE_RANK** can be in a view or a union. The **DENSE_RANK** function cannot be used in a subquery, a HAVING clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

LAG Function [Analytical]

An inter-row function that returns the value of an attribute in a previous row in the table or table partition.

Syntax

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

Parameters

Parameter	Description
value_expr	Table column or expression defining the offset data to return from the table.
offset	The number of rows above the current row, expressed as a non-negative exact numeric literal, or as a SQL variable with exact numeric data. The permitted range is 0 to 231.
default	The value to return if the *offset* value goes beyond the scope of the cardinality of the table or partition.

Parameter	Description
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Usage

The **LAG** function requires an **OVER** (**ORDER_BY**) window specification. The window partitioning clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause must not contain a window frame **ROWS**/**RANGE** specification.

You cannot define an analytic expression in *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

You must enter a non-negative numeric data type for *offset*. Entering **0** returns the current row. Entering a negative number generates an error.

The default value of *default* is **NULL**. The data type of *default* must be implicitly convertible to the data type of the *value_expr* value or else Sybase IQ generates a conversion error.

Example

The following example returns salary data from the Employees table, partitions the data by department ID, and orders the data according to employee start date. The **LAG** function returns the salary from the previous row (a physical offset of one row) and displays it under the **LAG (Salary)** column:

```
SELECT DepartmentID dID, StartDate, Salary, LAG(Salary, 1)
OVER(PARTITION BY dID ORDER BY StartDate) FROM Employees ORDER BY
1,2;
```

The returned result set is:

```
dID        StartDate    Salary      Lag(Salary)
=========  ===========  ==========  =============
100        1984-08-28   45,700.000  NULL
100        1985-01-01   62,000.000  45,700.000
100        1985-06-17   57,490.000  62,000.000
100        1986-06-07   72,995.000  57,490.000
100        1986-07-01   48,023.690  72,995.000
...
200        1985-02-03   38,500.000  NULL
200        1985-12-06   54,800.000  38,500.000
200        1987-02-19   39,300.000  54,800.000
200        1987-07-10   49,500.000  39,300.000
200        1988-10-04   54,600.000  49,500.000
200        1988-11-12   39,800.000  54,600.000
...
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

LEAD Function [Analytical]

An inter-row function that returns the value of an attribute in a subsequent row in the table or table partition.

Syntax

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

Parameters

Parameter	Description
value_expr	Table column or expression defining the offset data to return from the table.
offset	The number of rows below the current row, expressed as a non-negative exact numeric literal, or as a SQL variable with exact numeric data. The permitted range is 0 to 231.
default	The value to return if the *offset* value goes beyond the scope of the table or partition.
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Usage

The **LEAD** function requires an **OVER** (**ORDER_BY**) window specification. The window partitioning clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause must not contain a window frame **ROWS**/**RANGE** specification.

You cannot define an analytic expression in *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

You must enter a non-negative numeric data type for *offset*. Entering **0** returns the current row. Entering a negative number generates an error.

The default value of *default* is **NULL**. The data type of *default* must be implicitly convertible to the data type of the *value_expr* value or else Sybase IQ generates a conversion error.

Example

The following example returns salary data from the Employees table, partitions the data by department ID, and orders the data according to employee start date. The **LEAD** function returns the salary from the next row (a physical offset of one row) and displays it under the **LEAD (Salary)** column:

```
SELECT DepartmentID dID, StartDate, Salary, LEAD(Salary, 1)
OVER(PARTITION BY dID ORDER BY StartDate) FROM  Employees ORDER BY
1,2;
```

The returned result set is:

```
dID        StartDate    Salary      Lead(Salary)
=========  ===========  ==========  =============
100        1984-08-28    45,700.000    62,000.000
100        1985-01-01    62,000.000    57,490.000
100        1985-06-17    57,490.000    72,995.000
100        1986-06-07    72,995.000    48,023.690
...
100        1990-08-19    54,900.000    NULL
200        1985-02-03    38,500.000    39,300.000
200        1987-02-19    39,300.000    49,500.000
200        1987-07-10    49,500.000    54,600.000
200        1988-11-28    46,200.000    34,892.000
200        1989-06-01    34,892.000    87,500.000
...
200        1993-08-12    47,653.000    NULL
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

NTILE Function [Analytical]

Distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket.

Syntax

```
NTILE ( expression1 )
OVER ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant integer from 1 to 32767, which specifies the number of buckets.

Parameter	Description
expression2	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following example uses the **NTILE** function to determine the sale status of car dealers. The dealers are divided into four groups based on the number of cars each dealer sold. The dealers with ntile = 1 are in the top 25% for car sales.

```
SELECT dealer_name, sales,
NTILE(4) OVER ( ORDER BY sales DESC )
FROM carSales;


dealer_name         sales       ntile
Boston              1000        1
Worcester           950         1
Providence          950         1
SF                  940         1
Lowell              900         2
Seattle             900         2
Natick              870         2
New Haven           850         2
Portland            800         3
Houston             780         3
Hartford            780         3
Dublin              750         3
Austin              650         4
Dallas              640         4
Dover               600         4
```

To find the top 10% of car dealers by sales, you specify **NTILE(10)** in the example **SELECT** statement. Similarly, to find the top 50% of car dealers by sales, specify **NTILE(2)**.

Usage

NTILE is a rank analytical function that distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket. You can divide a result set into one-hundredths (percentiles), tenths (deciles), fourths (quartiles), or other numbers of groupings.

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

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

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

NTILE is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **NTILE** can be in a view or a union. The **NTILE** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one **NTILE** function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere

PERCENT_RANK Function [Analytical]

Computes the (fractional) position of one row returned from a query with respect to the other rows returned by the query, as defined by the **ORDER BY** clause.

Returns a decimal value between 0 and 1.

Syntax

PERCENT_RANK ( ) **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

The **PERCENT_RANK** function returns a DOUBLE value between 0 and 1.

Example

The following statement illustrates the use of the **PERCENT_RANK** function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       percent_rank_all
supplier#011     200000            0
supplier#002     200000            0
supplier#013     123000            0.3333
supplier#004     110000            0.5
supplier#035     110000            0.5
```

```
supplier#006     50000             0.8333
supplier#021     10000             1
```

Usage

PERCENT_RANK is a rank analytical function. The percent rank of a row R is defined as the rank of a row in the groups specified in the **OVER** clause minus one divided by the number of total rows in the groups specified in the **OVER** clause minus one. **PERCENT_RANK** returns a value between 0 and 1. The first row has a percent rank of zero.

The **PERCENT_RANK** of a row is calculated as

```
(Rx - 1) / (NtotalRow - 1)
```

where *Rx* is the rank position of a row in the group and *NtotalRow* is the total number of rows in the group specified by the **OVER** clause.

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

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

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

PERCENT_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **PERCENT_RANK** can be in a view or a union. The **PERCENT_RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

PERCENTILE_CONT Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a continuous distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

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

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, a “wrong argument for percentile” error is returned. If the argument value is less than 0 or greater than 1, a “data value out of range” error is returned
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the **PERCENTILE_CONT** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

The following **SELECT** statement contains the **PERCENTILE_CONT** function:

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

```
region          percentile_cont
Northeast       840
Northwest       740
South           470
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function **PERCENTILE_CONT** takes a percentile value as the function argument, and operates on a group of data specified in the **WITHIN GROUP** clause, or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. The data type of the **ORDER BY** expression for **PERCENTILE_CONT** must be numeric.

PERCENTILE_CONT requires a **WITHIN GROUP (ORDER BY)** clause.

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

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

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

The **PERCENTILE_CONT** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_CONT** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_CONT** function ignores the NULL value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

PERCENTILE_DISC Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a discrete distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

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

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a "wrong argument for percentile" error is returned. If the argument value is less than 0 or greater than 1, then a "data value out of range" error is returned.
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the **PERCENTILE_DISC** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales       region       dealer_name
900         Northeast    Boston
800         Northeast    Worcester
800         Northeast    Providence
700         Northeast    Lowell
540         Northeast    Natick
500         Northeast    New Haven
450         Northeast    Hartford
800         Northwest    SF
600         Northwest    Seattle
500         Northwest    Portland
400         Northwest    Dublin
500         South        Houston
400         South        Austin
300         South        Dallas
200         South        Dover
```

The following **SELECT** statement contains the **PERCENTILE_DISC** function:

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

```
region           percentile_cont
Northeast        900
Northwest        800
South            500
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function **PERCENTILE_DISC** takes a percentile value as the function argument and operates on a group of data specified in the **WITHIN GROUP** clause or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. **PERCENTILE_DISC** supports all data types that can be sorted in Sybase IQ.

PERCENTILE_DISC requires a **WITHIN GROUP (ORDER BY)** clause.

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

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

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

The **PERCENTILE_DISC** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_DISC** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_DISC** function ignores the NULL value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

RANK Function [Analytical]

Ranks items in a group.

Syntax

RANK ( ) **OVER** ([**PARTITION BY**] **ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

INTEGER

Example

This statement illustrates the use of the **RANK** function:

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

The results from the above query:

```
Surname           Sex      Salary     RANK
-------           ---      ------     ----
Savarino           F        72300.000  1
Jordan             F        51432.000  2
Clark              F        45000.000  3
Coleman            M        42300.000  1
Overbey            M        39300.000  2
```

Usage

RANK is a rank analytical function. The rank of row R is defined as the number of rows that precede R and are not peers of R. If two or more rows are not distinct within the groups specified in the **OVER** clause or distinct over the entire result set, then there are one or more gaps in the sequential rank numbering. The difference between **RANK** and **DENSE_RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

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

The **PARTITION BY** window partitioning clause in the **OVER (ORDER BY)** clause is optional.

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

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

RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **RANK** can be in a view or a union. The **RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATEADD Function [Date and Time]

Returns the date produced by adding the specified number of the specified date parts to a date.

Syntax

```
DATEADD ( date-part, numeric-expression, date-expression )
```

Parameters

Parameter	Description
date part	The date part to be added to the date.
numeric-expression	The number of date parts to be added to the date. The numeric-expression can be any numeric type; the value is truncated to an integer. The maximum microsecond in numeric-expression is 2147483647, that is, 35:47.483647 (35 mins 47 secs 483647 mcs).
date-expression	The date to be modified.

Returns

TIMESTAMP

Example

The following statement returns the value 1995-11-02 00:00:00.000:

```
SELECT DATEADD( MONTH, 102, '1987/05/02' ) FROM iq_dummy
```

The following statement returns the value 2009-11-10 14:57:52.722016:

```
SELECT DATEADD(MICROSECOND, 15, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 1985-05-02 00:00:00.123456:

```
SELECT DATEADD(MICROSECOND, 123456, '1985/05/02')
FROM iq_dummy
```

The following statement returns the value 1985-05-01 23:59:59.876544:

```
SELECT DATEADD(MICROSECOND, -123456, '1985/05/02')
FROM iq_dummy
```

The following statement returns the value 2009-11-03 11:10:42.033192:

```
SELECT DATEADD(MCS, 2, '2009-11-03 11:10:42.033190')
FROM iq_dummy
```

Usage

DATEADD is a Transact-SQL compatible data manipulation function.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DATECEILING Function [Date and Time]

Calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value of the specified granularity.

Syntax

DATECEILING (*date-part*, *datetime-expression [,multiple -expression]*)

Parameters

Parameter	Description
date-part	The date part to be added to the date.
datetime-expression	The date, time, or date-time expression containing the value you are evaluating.

Parameter	Description
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date-part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals. If multiple-expression evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified date-part, Sybase IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Examples

This statement returns the value August 13, 2009 10:40.00.000AM:

```
SELECT DATECEILING( MI, 'August 13, 2009, 10:32.00.132AM', 10) FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.456800 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM', 200 )
FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.600000 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM',
200000 ) FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.456789 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM') FROM
iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function increases the date and time up to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATECEILING**:

- DayofYear
- WeekDay

- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, Sybase IQ applies two-minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000
32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If Sybase IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATEDIFF Function [Date and Time]

Returns the interval between two dates.

Syntax

DATEDIFF (*date-part*, *date-expression1*, *date-expression2*)

Parameters

Table 91. Parameters

Parameter	Description
date-part	Specifies the date part in which the interval is to be measured.
date-expression1	The starting date for the interval. This value is subtracted from date-expression2 to return the number of date parts between the two arguments.
date-expression2	The ending date for the interval. *date-expression1* is subtracted from this value to return the number of date parts between the two arguments.

Returns

INT

Examples

The following statement returns 1:

```
SELECT DATEDIFF( HOUR, '4:00AM', '5:50AM' )
FROM iq_dummy
```

The following statement returns 102:

```
SELECT DATEDIFF( MONTH, '1987/05/02', '1995/11/15' )
FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( DAY, '00:00', '23:59' ) FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF( DAY, '1999/07/19 00:00', '1999/07/23
23:59' ) FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( MONTH, '1999/07/19', '1999/07/23' )
FROM iq_dummy
```

The following statement returns 1:

```
SELECT DATEDIFF( MONTH, '1999/07/19', '1999/08/23' )
FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF(MCS, '2009-11-03 11:10:42.033185',
'2009-11-03 11:10:42.033189') FROM iq_dummy
```

The following statement returns 15:

```
SELECT DATEDIFF(MICROSECOND, '2009-11-10
14:57:52.722001', '2009-11-10 14:57:52.722016')
FROM iq_dummy
```

The following statement returns 1,500,000:

```
SELECT DATEDIFF(MCS, '2000/07/07/07 07:07:06.277777',
'2000/07/07/07 07:07:07.777777') FROM iq_dummy
```

Usage

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to **(date2 - date1)**, in date parts.

DATEDIFF results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use **day** as the date part, **DATEDIFF** returns the number of midnights between the two times specified, including the second date, but not the first. For example, the following statement returns the value 5. Midnight of the first day 2003/08/03 is not included in the result. Midnight of the second day *is* included, even though the time specified is before midnight.

```
SELECT DATEDIFF( DAY, '2003/08/03 14:00', '2003/08/08 14:00' ) FROM
iq_dummy
```

When you use **month** as the date part, **DATEDIFF** returns the number of first-of-the-months between two dates, including the second date but not the first. For example, both of the following statements return the value 9:

```
SELECT DATEDIFF( MONTH, '2003/02/01', '2003/11/15' ) FROM iq_dummy;
SELECT DATEDIFF( MONTH, '2003/02/01', '2003/11/01' ) FROM iq_dummy;
```

The first date 2003/02/01 is a first-of-month, but is not included in the result of either query. The second date 2003/11/01 in the second query is also a first-of-month and *is* included in the result.

When you use **week** as the date part, **DATEDIFF** returns the number of Sundays between the two dates, including the second date but not the first. For example, in the month 2003/08, the dates of the Sundays are 03, 10, 17, 24, and 31. The following query returns the value 4:

```
SELECT DATEDIFF( week, '2003/08/03', '2003/08/31' ) FROM iq_dummy;
```

The first Sunday (2003/08/03) is not included in the result.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DATEFLOOR Function [Date and Time]

Calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value of the specified multiple with the specified granularity.

Syntax

DATEFLOOR (*date-part*, *datetime-expression [,multiple-expression]*)

Parameters

Parameter	Description
date part	The date part to be added to the date.
datetime-expression	The date, time, or date-time expression containing the value you are evaluating.
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by date-part to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals If multiple-expression evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified date-part, Sybase IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Examples

- This statement returns the value August 13, 2009 10:35:00.000AM:

  ```
  SELECT DATEFLOOR( MINUTE, 'August 13, 2009 10:35:22.123AM') FROM
  iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.456600 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM', 200 )
  FROM iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.400000 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM',
  200000 ) FROM iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.456789 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM') FROM
  iq_dummy
  ```

Usage

This function calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function reduces the date and time down to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATEFLOOR**:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, Sybase IQ applies two minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000
32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If Sybase IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATEFORMAT Function [Date and Time]

Returns a string representing a date expression in the specified format.

Syntax

```
DATEFORMAT ( datetime-expression, string-expression )
```

Parameters

Table 92. Parameters

Parameter	Description
datetime-expression	The date/time to be converted. Must be a date, time, timestamp, or character string.
string-expression	The format of the converted date.

Returns

VARCHAR

Example

The following statement returns string values like “Jan 01, 1989”:

```
SELECT DATEFORMAT( start_date, 'Mmm dd, yyyy' ) from Employees;
```

The following statement returns the string “Feb 19, 1987”:

```
SELECT DATEFORMAT( CAST ( ‘1987/02/19’ AS DATE ), ‘Mmm Dd, yyyy’ )
FROM iq_dummy
```

Usage

The *datetime-expression* to convert must be a date, time, or timestamp data type, but can also be a `CHAR` or `VARCHAR` character string. If the date is a character string, Sybase IQ implicitly converts the character string to date, time, or timestamp data type, so an explicit cast, as in the example above, is unnecessary.

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

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

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

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

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

To set the format used for dates retrieved from the database, see *Reference: Statements and Options > Database Options > Alphabetical List of Options > DATE_FORMAT Option*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATENAME Function [Date and Time]

Returns the name of the specified part (such as the month "June") of a date/time value, as a character string.

Syntax

```
DATENAME ( date-part, date-expression )
```

Parameters

Table 93. Parameters

Parameter	Description
date-part	The date part to be named.
date-expression	The date for which the date part name is to be returned. The date must contain the requested date-part.

Returns

VARCHAR

Example

The following statement returns the value May:

```
SELECT DATENAME( MONTH , '1987/05/02' )
FROM iq_dummy
```

The following statement returns the value 722,001:

```
SELECT DATENAME(MICROSECOND, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 777,777:

```
SELECT DATENAME(MICROSECOND, '2000/07/07
07:07:07.777777') FROM iq_dummy
```

The following statement returns the value 33,189:

```
SELECT DATENAME(MCS, '2009-11-03 11:10:42.033189')
FROM iq_dummy
```

Usage

DATENAME returns a character string, even if the result is numeric, such as 23, for the day.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DATEPART Function [Date and Time]

Returns an integer value for the specified part of a date/time value.

Syntax

```
DATEPART ( date-part, date-expression )
```

Parameters

Table 94. Parameters

Parameter	Description
date-part	The date part to be returned.
date-expression	The date for which the part is to be returned. The date must contain the date-part field.

Returns

INT

Example

The following statement returns the value 5:

```
SELECT DATEPART( MONTH, '1987/05/02' )
FROM iq_dummy
```

The following statement returns the value 722,001:

```
SELECT DATEPART(MICROSECOND, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 777,777:

```
SELECT DATEPART(MICROSECOND, '2000/07/07
07:07:07.777777') FROM iq_dummy
```

The following statement returns the value 33,189:

```
SELECT DATEPART(MCS, '2009-11-03 11:10:42.033189')
FROM iq_dummy
```

Usage

The **DATE**, **TIME**, and **DTTM** indexes do not support some date parts (Calyearofweek, Calweekofyear, Caldayofweek, Dayofyear, Millisecond, Microsecond).

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DATEROUND Function [Date and Time]

Calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest multiple of the specified value with the specified granularity.

Syntax

```
DATEROUND (date-part, datetime-expression [,multiple-expression] )
```

Parameters

Parameter	Description
date part	The date part to be returned.
datetime-expression	he date, time, or date-time expression containing the value you are evaluating.
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by date-part to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals. If *multiple-expression* evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified *date-part*, then Sybase IQ generates an error. If *multiple-expression* evaluates to a NULL, then the function result is NULL.

Examples

This statement returns the value August 13, 2009, 10:30.000AM:

```
SELECT DATEROUND( MI, 'August 13, 2009 10:33.123AM', 10) FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456600 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456500AM', 200 )
FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456789 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456789AM') FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456400 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456499AM', 200 )
FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function rounds the date and time to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATEROUND**:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, Sybase IQ applies two minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000
32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If Sybase IQ rounds to a multiple of the week date part, then the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATETIME Function [Date and Time]

Converts an expression into a timestamp.

Syntax

```
DATETIME ( expression )
```

Parameters

Table 95. Parameters

Parameter	Description
expression	The expression to be converted. The expression is usually a string. Conversion errors may be reported.

Returns

TIMESTAMP

Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DAYNAME Function [Date and Time]

Returns the name of the day of the week from the specified date.

Syntax

```
DAYNAME ( date-expression )
```

Parameters

Table 96. Parameters

Parameter	Description
date-expression	The date.

Returns

VARCHAR

Example

The following statement returns the value Saturday:

```
SELECT DAYNAME ( '1987/05/02' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

DAYS Function [Date and Time]

Returns the number of days since an arbitrary starting date, returns the number of days between two specified dates, or adds the specified *integer-expression* number of days to a given date.

DAYS ignores hours, minutes, and seconds.

Syntax

```
DAYS ( datetime-expression )
|   ( datetime-expression, datetime-expression )
|   ( datetime-expression, integer-expression )
```

Parameters

Table 97.

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of days to be added to the datetime-expression. If the integer-expression is negative, the appropriate number of days are subtracted from the date/time. If you supply an integer expression, the datetime-expression must be explicitly cast as a date.

Returns

INT when you specify two datetime expressions.

TIMESTAMP when the second argument you specify is an integer.

Examples

The following statement returns the integer value 729948:

```
SELECT DAYS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the integer value -366, which is the difference between the two dates:

```
SELECT DAYS( '1998-07-13 06:07:12',
'1997-07-12 10:07:12' ) FROM iq_dummy
```

The following statement returns the value 1999-07-14:

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 )
FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

DOW Function [Date and Time]

Returns a number from 1 to 7 representing the day of the week of the specified date, with Sunday=1, Monday=2, and so on.

Syntax

```
DOW ( date-expression )
```

Parameters

Table 98. Parameters

Parameter	Description
date-expression	The date.

Returns

SMALLINT

Example

The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' ) FROM iq_dummy
```

Usage

See *Reference: Statements and Options > Alphabetical List of Options > DATE_FIRST_DAY_OF_WEEK Option* if you need Monday (or another day) to be the first day of the week.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

GETDATE Function [Date and Time]

Returns the current date and time.

Syntax

```
GETDATE ()
```

Returns

TIMESTAMP

Example

The following statement returns the system date and time.

```
SELECT GETDATE( ) FROM iq_dummy
```

Usage

GETDATE is a Transact-SQL compatible data manipulation function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

HOURS Function [Date and Time]

Returns the number of hours since an arbitrary starting date and time, the number of whole hours between two specified times, or adds the specified integer-expression number of hours to a time.

Syntax

```
HOURS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Table 99. Parameters

Parameter	Description
datetime-expression	A date and time.

Parameter	Description
integer-expression	The number of hours to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of hours are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

Returns

INT

Examples

The following statement returns the value 17518758:

```
SELECT HOURS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 4, to signify the difference between the two times:

```
SELECT HOURS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-13 02:05:07.000:

```
SELECT HOURS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5 ) FROM iq_dummy
```

Usage

The second syntax returns the number of whole hours from the first date/time to the second date/time. The number might be negative.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

MINUTE Function [Date and Time]

Returns a number from 0 to 59 corresponding to the minute component of the specified date/time value.

Syntax

MINUTE (*datetime-expression*)

Parameters

Parameter	Description
datetime-expression	The date/time value.

Returns

SMALLINT

Example

The following statement returns the value 22:

```
SELECT MINUTE( '1998-07-13 12:22:34' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

MINUTES Function [Date and Time]

Returns the number of minutes since an arbitrary date and time, the number of whole minutes between two specified times, or adds the specified integer-expression number of minutes to a time.

Syntax

```
MINUTES ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of minutes to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of minutes are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `datetime` data type

Returns

INT

TIMESTAMP

Examples

Returns the value 1051125487:

```
SELECT MINUTES( '1998-07-13 06:07:12' ) FROM iq_dummy
```

Returns the value 240, to signify the difference between the two times:

```
SELECT MINUTES( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

Returns the datetime value 1999-05-12 21:10:07.000:

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole minutes from the first date/time to the second date/time. The number might be negative.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise.

MONTH Function [Date and Time]

Returns a number from 1 to 12 corresponding to the month of the given date.

Syntax

```
MONTH ( date-expression )
```

Parameters

Parameters	Description
date-expression	A date/time value.

Returns

SMALLINT

Example

The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise.

MONTHNAME Function [Date and Time]

Returns the name of the month from the specified date expression.

Syntax

```
MONTHNAME ( date-expression )
```

Parameters

Parameter	Description
date-expression	The datetime value.

Returns

VARCHAR

Example

The following statement returns the value **September**, when the `DATE_ORDER` option is set to the default value of *ymd*.

```
SELECT MONTHNAME( '1998-09-05' ) FROM iq_dummy
```

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > DATE_ORDER Option*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

MONTHS Function [Date and Time]

Returns the number of months since an arbitrary starting date/time or the number of months between two specified date/times, or adds the specified integer-expression number of months to a date/time.

Syntax

```
MONTHS ( date-expression
| date-expression, datetime-expression
| date-expression, integer-expression )
```

Parameters

Parameter	Description
date-expression	A date and time.

Parameter	Description
integer-expression	The number of months to be added to the *date-expression.* If *integer-expression* is negative, the appropriate number of months are subtracted from the date/time value. If you supply an integer expression, the *date-expression* must be explicitly cast as a `datetime` data type.

Returns

INT

TIMESTAMP

Examples

The following statement returns the value 23982:

```
SELECT MONTHS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates:

```
SELECT MONTHS( '1999-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-10-12 21:05:07.000:

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The first syntax returns the number of months since an arbitrary starting date. This number is often useful for determining whether two date/time expressions are in the same month in the same year.

```
MONTHS( invoice_sent ) = MONTHS( payment_received )
```

Comparing the **MONTH** function would incorrectly include a payment made 12 months after the invoice was sent.

The second syntax returns the number of months from the first date to the second date. The number might be negative. It is calculated from the number of the first days of the month between the two dates. Hours, minutes and seconds are ignored.

The third syntax adds *integer-expression* months to the given date. If the new date is past the end of the month (such as **MONTHS** ('1992-01-31', 1)) the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of months are subtracted from the date. Hours, minutes and seconds are ignored.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

NOW Function [Date and Time]

Returns the current date and time. This is the historical syntax for **CURRENT TIMESTAMP**.

Syntax

```
NOW ( * )
```

Returns

TIMESTAMP

Example

The following statement returns the current date and time.

```
SELECT NOW(*) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

QUARTER Function [Date and Time]

Returns a number indicating the quarter of the year from the supplied date expression.

Syntax

```
QUARTER( date-expression )
```

Parameters

Parameter	Description
date-expression	A date.

Returns

INT

Example

With the **DATE_ORDER** option set to the default of *ymd*, the following statement returns the value 2:

```
SELECT QUARTER ( '1987/05/02' ) FROM iq_dummy
```

See *Reference: Statements and Options > Database Options > Alphabetical List of Options > DATE_ORDER Option*.

Usage
This table lists the dates in the quarters of the year.

Table 100. Values of quarter of the year

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30
4	October 1 to December 31

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SECOND Function [Date and Time]

Returns a number from 0 to 59 corresponding to the second component of the given date/time value.

Syntax

```
SECOND ( datetime-expression )
```

Parameters

Parameter	Description
datetime-expression	The date/time value.

Returns

SMALLINT

Example
The following statement returns the value 5:

```
SELECT SECOND( '1998-07-13 08:21:05' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Compatible with Adaptive Server Enterprise.

SECONDS Function [Date and Time]

Returns the number of seconds since an arbitrary starting date and time, the number of seconds between two times, or adds an integer amount of seconds to a time.

Syntax

```
SECONDS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of seconds to be added to the date-time-expression. If integer-expression is negative, the appropriate number of minutes are subtracted from the date/time value. If you supply an integer expression, the datetime-expression must be explicitly cast as a datetime data type.

Returns

INTEGER

TIMESTAMP

Examples

The following statement returns the value 3600:

```
SELECT ( SECONDS( '1998-07-13 06:07:12' ) -
SECONDS( '1998-07-13 05:07:12' )) FROM iq_dummy
```

The following statement returns the value 14400, to signify the difference between the two times:

```
SELECT SECONDS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-12 21:05:12.000:

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole seconds from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

TODAY Function [Date and time]

Returns the current date. This is the historical syntax for **CURRENT DATE**.

Syntax

```
TODAY ( * )
```

Returns

DATE

Example

The following statement returns the current day according to the system clock.

```
SELECT TODAY( * ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

WEEKS Function [Date and Time]

Returns the number of weeks since an arbitrary starting date/time, returns the number of weeks between two specified date/times, or adds the specified integer-expression number of weeks to a date/time.

Syntax

```
WEEKS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.

Parameter	Description
integer-expression	The number of weeks to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of weeks are subtracted from the date/time value. Hours, minutes, and seconds are ignored. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `DATETIME` data type.

Returns

Syntax 1 returns an INTEGER.

Syntax 2 returns a TIMESTAMP.

Examples

The following statement returns the value 104278:

```
SELECT WEEKS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 9, to signify the difference between the two dates:

```
SELECT WEEKS( '1999-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the timestamp value 1999-06-16 21:05:07.000:

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

Weeks are defined as going from Sunday to Saturday, as they do in a North American calendar. The number returned by the first syntax is often useful for determining if two dates are in the same week.

```
WEEKS ( invoice_sent ) = WEEKS ( payment_received ) FROM iq_dummy
```

In the second syntax, the value of **WEEKS** is calculated from the number of Sundays between the two dates. Hours, minutes, and seconds are ignored. This function is not affected by the `DATE_FIRST_DAY_OF_WEEK` option.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

YEAR Function [Date and Time]

Returns a 4-digit number corresponding to the year of the given date/time.

Syntax

```
YEAR ( datetime-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.

Returns

SMALLINT

Example

The following statement returns the value 1998:

```
SELECT YEAR( '1998-07-13 06:07:12' ) FROM iq_dummy
```

Usage

The **YEAR** function is the same as the first syntax of the **YEARS** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

YEARS Function [Date and Time]

Returns a 4-digit number corresponding to the year of a given date/time, returns the number of years between two specified date/times, or adds the specified integer-expression number of years to a date/time.

Syntax

```
YEARS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expressio	A date and time.

Parameter	Description
integer-expression	The number of years to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of years are subtracted from the datetime value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `DATETIME` data type.

Returns

Syntax 1 returns an INTEGER.

Syntax 2 returns a TIMESTAMP.

Examples

The following statement returns the value 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates.

```
SELECT YEARS( '1997-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the YEARS(cast('1999-05-12 21:05:07' as timestamp), 5) value 2004-05-12 21:05:07.000:

```
SELECT YEARS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The first syntax of the **YEARS** function is the same as the **YEAR** function.

The second syntax returns the number of years from the first date to the second date, calculated from the number of first days of the year between the two dates. The number might be negative. Hours, minutes, and seconds are ignored. For example, the following statement returns 2, which is the number of first days of the year between the specified dates:

```
SELECT YEARS ( '2000-02-24', '2002-02-24' ) FROM iq_dummy
```

The next statement also returns 2, even though the difference between the specified dates is not two full calendar years. The value 2 is the number of first days of the year (in this case January 01, 2001 and January 01, 2002) between the two dates.

```
SELECT YEARS ( '2000-02-24', '2002-02-20' ) FROM iq_dummy
```

The third syntax adds an *integer-expression* number of years to the given date. If the new date is past the end of the month (such as **SELECT YEARS** (**CAST** (‘1992-02-29’ AS **TIMESTAMP**), 1)), the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of years is subtracted from the date. Hours, minutes, and seconds are ignored.

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

YMD Function [Date and Time]

Returns a date value corresponding to the given year, month, and day of the month.

Syntax

```
YMD ( integer-expression1, integer-expression2, integer-
expression3 )
```

Parameters

Parameter	Description
integer-expression1	The year.
integer-expression2	The number of the month. If the month is outside the range 1–12, the year is adjusted accordingly.
integer-expression3	The day number. The day is allowed to be any integer, the date is adjusted accordingly.

Returns

DATE

Examples

The following statement returns the value 1998-06-12:

```
SELECT YMD( 1998, 06, 12 ) FROM iq_dummy
```

If the values are outside their normal range, the date adjusts accordingly. For example, the following statement returns the value 1993-03-01:

```
SELECT YMD( 1992, 15, 1 ) FROM iq_dummy
```

The following statement returns the value 1993-02-28:

```
SELECT YMD ( 1992, 15, 1-1 ) FROM iq_dummy
```

The following statement returns the value 1992-02-29:

```
SELECT YMD ( 1992, 3, 1-1 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

AVG Function [Aggregate]

Computes the average of a numeric expression for a set of rows, or computes the average of a set of unique values.

Syntax

```
AVG ( numeric-expression  | DISTINCT column-name )
```

Parameters

Table 101. Parameters

Parameter	Description
numeric-expression	The value whose average is calculated over a set of rows.
DISTINCT column-name	Computes the average of the unique values in column-name. This is of limited usefulness, but is included for completeness

Returns

Returns the NULL value for a group containing no rows.

Returns DOUBLE if the argument is DOUBLE, otherwise NUMERIC.

Example

The following statement returns the value 49988.6:

```
SELECT AVG ( salary ) FROM Employees
```

Usage

This average does not include rows where *numeric -expression* is the NULL value. Returns the NULL value for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

CORR Function [Aggregate]

Returns the correlation coefficient of a set of number pairs.

Syntax 1

```
CORR (dependent-expression, independent-expression)
```

Syntax 2

```
CORR (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Table 102. Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent-expression.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Example

The following example performs a correlation to discover whether age is associated with income level. This function returns the value 0.440227:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM
Employees;
```

Usage

The **CORR** function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then CORR returns NULL.

dependent-expression and independent-expression are both numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL. The following computation is made:

COVAR_POP (y, x) / (STDDEV_POP (x) * STDDEV_POP (y))

where x represents the dependent-expression and y represents the independent-expression.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

COVAR_POP Function [Aggregate]

Returns the population covariance of a set of number pairs.

Syntax 1

```
COVAR_POP (dependent-expression, independent-expression)
```

Syntax 2

```
COVAR_POP (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Table 103. Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 73785.840059:

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **COVAR_POP** returns NULL.

dependent-expression and independent-expression are both numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL. The following computation is made:

(SUM(x*y) - SUM(x) * SUM(y) / n) / n

where x represents the dependent-expression and y represents the independent-expression.

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

COVAR_SAMP Function [Aggregate]

Returns the sample covariance of a set of number pairs.

Syntax 1

```
COVAR_SAMP (dependent-expression, independent-expression)
```

Syntax 2

```
COVAR_SAMP (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Table 104. Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 74782.946005:

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) ) FROM
Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **COVAR_SAMP** returns NULL.

Both dependent-expression and independent-expression are numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL.

```
(SUM(x*y) - SUM(x) * SUM(y) / n) / (n-1)
```

where x represents the dependent-expression and y represents the independent-expression.

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions.*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

COUNT Function [Aggregate]

Counts the number of rows in a group, depending on the specified parameters.

Syntax

```
COUNT ( * | expression | DISTINCT column-name )
```

Parameters

Parameter	Description
*	Returns the number of rows in each group.
expression	Returns the number of rows in each group where expression is not the NULL value.
DISTINCT column-name	Returns the number of different values in column-name. Rows where the value is the NULL value are not included in the count.

Note: When the query results are displayed, the * is not displayed in the column header, and appears as:

```
Count()
```

Returns

INT

Example

Returns each unique city, and the number of rows with that city value:

```
SELECT city , Count(*)
FROM Employees
GROUP BY city
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

EXP_WEIGHTED_AVG Function [Aggregate]

Calculates an exponential weighted moving average.

Weightings determine the relative importance of each quantity that makes up the average.

Syntax

EXP_WEIGHTED_AVG (*expression*, *period-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
expression	A numeric expression for which a weighted value is being computed.
period-expression	A numeric expression specifying the period for which the average is to be computed.

Usage

Similar to the **WEIGHTED_AVG** function, the weights in **EXP_WEIGHTED_AVG** decrease over time. However, weights in WEIGHTED_AVG decrease arithmetically, whereas weights in EXP_WEIGHTED_AVG decrease exponentially. Exponential weighting applies more weight to the most recent values, and decreases the weight for older values while still applying some weight.

Sybase IQ calculates the exponential moving average using:

S*C+(1-S)*PEMA

In the calculation above, Sybase IQ applies the smoothing factor by multiplying the current closing price (C) by the smoothing constant (S) added to the product of the previous day's exponential moving average value (PEMA) and 1 minus the smoothing factor.

Sybase IQ calculates the exponential moving average over the entire period specified by the **OVER** clause. *period-expression* specifies the moving range of the exponential moving average.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement. The *window-spec* must contain an **ORDER BY** statement and cannot contain a frame specification.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause. DISTINCT is not supported.

Example

The following example returns an exponential weighted average of salaries for employees in Florida with the salary of recently hired employees contributing the most weight to the average. There are three rows used in the weighting:

```
SELECT DepartmentID, Surname, Salary,EXP_WEIGHTED_AVG(Salary, 3)
OVER (ORDER BY YEAR(StartDate) DESC) as "W_AVG"FROM EmployeesWHERE
State IN ('FL') ORDER BY StartDate DESC
```

The returned result set is:

Table 105. EXP_WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
400	Evans	68,940.000	34,470.000000
300	Litton	58,930.000	46,700.000000
200	Sterling	64,900.000	55,800.000000
200	Kelly	87,500.000	71,650.000000
400	Charlton	28,300.000	49,975.000000
100	Lull	87,900.000	68,937.500000
100	Gowda	59,840.000	60,621.875000
400	Francis	53,870.000	61,403.750000

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

FIRST_VALUE Function [Aggregate]

Returns the first value from a set of values.

Syntax

```
FIRST_VALUE (expression [IGNORE NULLS | RESPECT NULLS])
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
expression	The expression on which to determine the first value in an ordered set.

Returns

Data type of the argument.

Usage

FIRST_VALUE returns the first value in a set of values, which is usually an ordered set. If the first value in the set is null, then the function returns NULL unless you specify **IGNORE NULLS**. If you specify IGNORE NULLS, then **FIRST_VALUE** returns the first non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use **FIRST_VALUE** or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an **ORDER BY** expression, or if the **ORDER BY** expression is not precise enough to guarantee a unique ordering, then the result is arbitrary. If there is no window-spec, then the result is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: DISTINCT is not supported.

Example

The following example returns the relationship, expressed as a percentage, between each employee’s salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
100 * Salary / ( FIRST_VALUE( Salary ) OVER (
PARTITION BY DepartmentID  ORDER BY Year(StartDate) DESC ) )
```

```
AS percentage
FROM Employees order by DepartmentID DESC;
```

The returned result set is:

Table 106. FIRST_VALUE result set

DepartmentID	EmployeeID	Percentage
500	1,658	100.00000000000000000000
500	1,570	138.842709713689113761394
500	1,615	110.428462434244870095972
500	1,013	109.585190539292454724330
500	750	137.734409508894510701521
500	921	167.449704854836766654619
500	868	113.239368750752921334778
500	703	222.867927558928643135365
500	191	119.664297474199895594908
400	1,684	100.00000000000000000000
400	1,740	76.128652163477274215016
400	1,751	76.353400685155687446813
400	1,607	133.758100765890593292456
400	1,507	77.996465120338650199655
400	1,576	150.428767810774836893669

In this example, employee 1658 is the first row for department 500, indicating that employee 1658 is the most recent hire in that department, and therefore receives a percentage of 100%. Percentages for the remaining employees in department 500 are calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139% of what employee 1658 earns.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

GROUPING Function [Aggregate]

Identifies whether a column in a **ROLLUP** or **CUBE** operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

Syntax

```
GROUPING ( group-by-expression )
```

Parameters

Parameter	Description
group-by-expression	An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause with the ROLLUP or CUBE keyword. The function identifies subtotal rows added to the result set by a ROLLUP or CUBE operation.

Currently, Sybase IQ does not support the **PERCENTILE_CONT** or **PERCENTILE_DISC** functions with **GROUP BY CUBE** operations.

Returns

Value	Description
1	Indicates that group-by-expression is NULL because it is part of a subtotal row. The column is not a prefix column for that row.
0	Indicates that group-by-expression is a prefix column of a subtotal row.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

LAST_VALUE Function [Aggregate]

Returns the last value from a set of values.

Syntax

```
LAST_VALUE (expression [IGNORE NULLS | RESPECT NULLS])
```

```
OVER (window-spec)
```

Parameters

Parameter	Definition
expression	The expression on which to determine the last value in an ordered set

Returns

Data type of the argument.

Usage

LAST_VALUE returns the last value in a set of values, which is usually an ordered set. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then **LAST_VALUE** returns the last non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use **LAST_VALUE** or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an **ORDER BY** expression, or if the **ORDER BY** expression is not precise enough to guarantee a unique ordering, then the result is arbitrary. If there is no window-spec, then the result is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: DISTINCT is not supported.

Example

The following example returns the salary of each employee, plus the name of the employee with the highest salary in their department:

```
SELECT GivenName + ' ' + Surname AS employee_name,
    Salary, DepartmentID,
    LAST_VALUE( employee_name ) OVER Salary_Window AS
highest_paid
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID ORDER BY Salary
    RANGE BETWEEN UNBOUNDED PRECEDING
    AND UNBOUNDED FOLLOWING )
ORDER BY DepartmentID DESC;
```

The returned result set is:

Table 107. LAST_VALUE result set

employee_name	Salary	DepartmentID	highest_paid
Michael Lynch	24,903.000	500	Jose Martinez
Joseph Barker	27,290.000	500	Jose Martinez
Sheila Romero	27,500.000	500	Jose Martinez
Felicia Kuo	28,200.000	500	Jose Martinez
Jeannette Bertrand	29,800.000	500	Jose Martinez
Jane Braun	34,300.000	500	Jose Martinez
Anthony Rebeiro	34,576.000	500	Jose Martinez
Charles Crowley	41,700.000	500	Jose Martinez
Jose Martinez	55,500.800	500	Jose Martinez
Doug Charlton	28,300.000	400	Scott Evans
Elizabeth Lambert	29,384.000	400	Scott Evans
Joyce Butterfield	34,011.000	400	Scott Evans
Robert Nielsen	34,889.000	400	Scott Evans
Alex Ahmed	34,992.000	400	Scott Evans
Ruth Wetherby	35,745.000	400	Scott Evans
...	...	...	...

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

MAX Function [Aggregate]

Returns the maximum *expression* value found in each group of rows.

Syntax

```
MAX ( expression
| DISTINCT column-name )
```

Parameters

Parameter	Description
expression	The expression for which the maximum value is to be calculated. This is commonly a column name.
DISTINCT column-name	Returns the same as MAX (expression), and is included for completeness.

Returns

The same data type as the argument.

Example

The following statement returns the value 138948.000, representing the maximum salary in the `Employees` table:

```
SELECT MAX ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

MEDIAN Function [Aggregate]

Returns the median of an expression.

Syntax 1

```
MEDIAN([ALL | DISTINCT] expression)
```

Syntax 2

```
MEDIAN([ALL | DISTINCT] expression)

OVER (window-spec)
```

Parameters

Parameter	Description
expression	A numeric expression for which a median value is to be computed.

Usage

The median is the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.

The data type of the returned value is the same as that of the input value. NULLs are ignored in the calculation of the median value. You can use the optional keyword **DISTINCT** to eliminate duplicate values before the aggregate function is applied. **ALL**, which performs the operation on all rows, is the default.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: The *window-spec* cannot contain a **ROW**, **RANGE** or **ORDER BY** specification; *window-spec* can only specify a **PARTITION** clause. DISTINCT is not supported if a **WINDOW** clause is used.

Example

The following query returns the median salary for each department in Florida:

```
SELECT DepartmentID, Surname, Salary,
MEDIAN(Salary) OVER (PARTITION BY DepartmentID) "Median"
FROM Employees
WHERE State IN ('FL')
```

The returned result is:

Table 108. MEDIAN result set

DepartmentID	Surname	Salary	Median
100	Lull	87,900.000	73,870.000
100	Gowda	59,840.000	73,870.000
200	Sterling	64,900.000	76,200.000
200	Kelly	87,500.000	76,200.000
300	Litton	58,930.000	58,930.000

DepartmentID	Surname	Salary	Median
400	Francis	53,870.000	38,70.000
400	Charlton	28,300.000	53,870.000
400	Evans	68,940.000	53,870.000

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

MIN Function [Aggregate]

Returns the minimum expression value found in each group of rows.

Syntax

```
MIN ( expression
| DISTINCT column-name )
```

Parameters

Parameter	Description
expression	The expression for which the minimum value is to be calculated. This is commonly a column name.
DISTINCT column-name	Returns the same as MIN (expression), and is included for completeness.

Returns

The same data type as the argument.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

REGR_AVGX Function [Aggregate]

Computes the average of the independent variable of the regression line.

Syntax 1

```
REGR_AVGX (dependent-expression, independent-expression)
```

Syntax 2

```
REGR_AVGX (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_AVGX** returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where x represents the independent-expression:

```
AVG (x)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example calculates the average of the dependent variable, employee age:

```
SELECT REGR_AVGX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_AVGY Function [Aggregate]

Computes the average of the dependent variable of the regression line.

Syntax 1

```
REGR_AVGY(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_AVGY(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_AVGY** returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression:

```
AVG(y)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example calculates the average of the independent variable, employee salary. This function returns the value 49988.6232:

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_COUNT Function [Aggregate]

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax 1

```
REGR_COUNT(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_COUNT(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

INTEGER

Usage

This function returns an UNSIGNED BIGINT as the result.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns a value that indicates the number of non-NULL pairs that were used to fit the regression line. This function returns the value 75:

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_INTERCEPT Function [Aggregate]

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax 1

REGR_INTERCEPT(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_INTERCEPT(*dependent-expression*, *independent-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, **REGR_INTERCEPT** returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
AVG(y) - REGR_SLOPE(y, x) * AVG(x)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 1874.5805688517603:

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_R2 Function [Aggregate]

Computes the coefficient of determination (also referred to as R-squared or the goodness-of-fit statistic) for the regression line.

Syntax 1

REGR_R2(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_R2(*dependent-expression*, *independent-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_R2** returns NULL.

REGR_R2 is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. Sybase IQ then applies the following algorithm:

- **REGR_R2** calculates VAR_POP(x) and returns NULL if VAR_POP(x) = 0; otherwise, it calculates VAR_POP(y) and returns the value 1 if VAR_POP(y) = 0.
- If neither VAR_POP(x) or VAR_POP(y) is zero, the return value is POWER(CORR(y,x), 2)

where y represents the *dependent-expression* and x represents the *independent-expression*.

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. **DISTINCT** is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 0.193799597103256 53:

```
SELECT REGR_R2( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.

- Sybase—Compatible with SQL Anywhere.

REGR_SLOPE Function [Aggregate]

Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

```
REGR_SLOPE(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SLOPE(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SLOPE** returns NULL.

REGR_SLOPE is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

```
COVAR_POP(x, y) / VAR_POP(y)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 935.3429749445614:

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_SXX Function [Aggregate]

Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

REGR_SXX(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_SXX(*dependent-expression*, *independent-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SXX** returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
REGR_COUNT(y, x) * VAR_POP(x)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 5916.4800000000105:

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_SXY Function [Aggregate]

Returns the sum of products of the dependent and independent variables. Use REGR_SXY to evaluate the statistical validity of a regression model.

Syntax 1

```
REGR_SXY(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SXY(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
REGR_COUNT(x, y) * COVAR_POP(x,
y)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 5533938.004400015.

```
SELECT REGR_SXY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_SYY Function [Aggregate]

Returns values that can evaluate the statistical validity of a regression model.

Syntax 1

REGR_SYY(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_SYY(*dependent-expression*, *independent-expression*)

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SYY** returns NULL.

The function is applied to the set of (*dependent-expression* and independent-expression) pairs after eliminating all pairs for which either *dependent-expression* or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the *dependent-expression* and x represents the independent-expression:

```
REGR_COUNT(x, y) * VAR_POP(y)
```

See *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions > Mathematical formulas for the aggregate functions*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 26, 708, 672,843.3002:

```
SELECT REGR_SYY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

STDDEV Function [Aggregate]

Returns the standard deviation of a set of numbers.

Syntax

```
STDDEV ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	Any numeric data type (FLOAT, REAL, or DOUBLE precision) expression.

Returns

DOUBLE

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

Salary
51432.000
57090.000
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00

The following statement returns the value 32617.8446712838471:

```
SELECT STDDEV ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE Name = 'Tee Shirt'
```

Name	UnitPrice
Tee Shirt	9.00
Tee Shirt	14.00
Tee Shirt	14.00

The following statement returns the value 2.88675134594813049:

```
SELECT STDDEV ( UnitPrice ) FROM Products
WHERE Name = 'Tee Shirt'
```

Usage

The formula used to calculate **STDDEV** is:

$$stddev = \sqrt{variance}$$

STDDEV returns a result of data type `DOUBLE` precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

STDDEV does not support the keyword DISTINCT. A syntax error is returned if you use DISTINCT with **STDDEV**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

STDDEV_POP Function [Aggregate]

Computes the standard deviation of a population consisting of a numeric-expression, as a `DOUBLE`.

Syntax

```
STDDEV_POP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose population-based standard deviation is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
  AS Quarter, AVG( quantity ) AS Average,
  STDDEV_POP ( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.2794
2000	2	27.050847	15.0270
...	...	...	...

Usage

Computes the population standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

STDDEV_SAMP Function [Aggregate]

Computes the standard deviation of a sample consisting of a numeric-expression, as a `DOUBLE`.

Syntax

STDDEV_SAMP ([ALL] *expression*)

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose sample-based standard deviation is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
  AS Quarter, AVG( quantity ) AS Average,
  STDDEV_SAMP( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.3218
2000	2	27.050847	15.0696
...	...	...	...

Usage

Note: STDDEV_SAMP is an alias for **STDDEV**.

Computes the sample standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

NULL returns NULL for a one-element input set.

Standard deviations are computed according to the following formula, which assumes a normal distribution:

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{(n-1)}}$$

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SUM Function [Aggregate]

Returns the total of the specified expression for each group of rows.

Syntax

SUM (*expression* | **DISTINCT** *column-name*)

Parameters

Parameter	Description
expression	The object to be summed. This is commonly a column name.
DISTINCT column-name	Computes the sum of the unique values in *column-name* for each group of rows. This is of limited usefulness, but is included for completeness.

Returns

INTEGER

DOUBLE

NUMERIC

Example

The following statement returns the value 3749146.740:

```
SELECT SUM( salary )
FROM Employees
```

Usage

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

VAR_SAMP Function [Aggregate]

Computes the statistical variance of a sample consisting of a numeric-expression, as a `DOUBLE`.

Syntax

```
VAR_SAMP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ShipDate ) AS Year, quarter( ShipDate )
  AS Quarter, AVG( Quantity ) AS Average,
  VAR_SAMP( Quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter
```

Year	Quarter	Average	Variance
2000	1	25.775148	205.1158
2000	2	27.050847	227.0939
...	...	...	...

Usage

Note: **VAR_SAMP** is an alias of **VARIANCE**.

Computes the sample variance of *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of *value expression*, from the mean of *value expression*, divided by one less than the number of rows (remaining) in the group or partition.

NULL returns NULL for a one-element input set in Sybase IQ 12.7 and later. In versions earlier than 12.7, NULL returned zero.

Variances are computed according to the following formula, which assumes a normal distribution:

$$\frac{\sum (x_i - \bar{x})^2}{n}$$

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

VARIANCE Function [Aggregate]

Returns the variance of a set of numbers.

Syntax

```
VARIANCE ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	Any numeric data type (`FLOAT`, `REAL`, or `DOUBLE`) expression. The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.

Returns

DOUBLE

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

salary
51432.000
57090.000
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00

salary
75400.00

The following statement returns the value 1063923790.99999994:

```
SELECT VARIANCE ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE name = 'Tee Shirt'
```

UnitPrice
9.00
14.00
14.00

The following statement returns the value 8.333333333333334327:

```
SELECT VARIANCE ( UnitPrice ) FROM Products
WHERE name = 'Tee Shirt'
```

Usage

The formula used to calculate **VARIANCE** is

$$var = \frac{n\sum x^2 - (\sum x)^2}{n(n-1)}$$

VARIANCE returns a result of data type `double-precision floating-point`. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

VARIANCE does not support the keyword DISTINCT. A syntax error is returned if DISTINCT is used with **VARIANCE**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

WEIGHTED_AVG Function [Aggregate]

Calculates an arithmetically (or linearly) weighted average.

A weighted average is an average in which each quantity to be averaged is assigned a weight. Weightings determine the relative importance of each quantity that make up the average.

Syntax

```
WEIGHTED_AVG (expression)
```

```
OVER (window-spec)
```

window-spec: See the Usage section, below.

Parameters

Parameter	Description
expression	A numeric expression for which a weighted value is being computed.

Usage

Use the **WEIGHTED_AVG** function to create a weighted moving average. In a weighted moving average, weights decrease arithmetically over time. Weights decrease from the highest weight for the most recent data points, down to zero.

Figure 3: WEIGHTED_AVG Calculation

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + \cdots + 2p_{M-n+2} + p_{M-n+1}}{n + (n-1) + \cdots + 2 + 1}$$

To exaggerate the weighting, you can average two or more weighted moving averages together, or use an **EXP_WEIGHTED_AVG** function instead.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

window-spec:

- Must contain an ORDER BY specifier.
- Cannot contain FOLLOWING or RANGE specifiers.
- The second argument of the ROW specifier—if provided—must be CURRENT ROW.
- Cannot contain NULL values.
- Cannot contain the DISTINCT specifier.
- UNBOUNDED PRECEDING is supported, but may result in poor performance if used

Example

The following example returns a weighted average of salaries by department for employees in Florida, with the salary of recently hired employees contributing the most weight to the average:

```
SELECT DepartmentID, Surname, Salary,
WEIGHTED_AVG(Salary) OVER (PARTITION BY DepartmentID
ORDER BY YEAR(StartDate) DESC) as "W_AVG"
FROM Employees
WHERE State IN ('FL') ORDER BY DepartmentID
```

The returned result set is:

Table 109. WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
100	Lull	87,900.000	87,900.000000
100	Gowda	59,840.000	69,193.333333
200	Sterling	64,900.000	64,900.000000
200	Kelly	87,500.000	79,966.666667
300	Litton	58,930.000	58,930.000000
400	Evans	68,940.000	68,940.000000
400	Charlton	28,300.000	41,846.666667
400	Francis	53,870.000	47,858.333333

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

AES_ENCRYPT Function [String]

Encrypts the specified values using the supplied encryption key, and returns a `VARBINARY` or `LONG VARBINARY`.

Syntax

```
AES_ENCRYPT( string-expression, key )
```

Parameters

string-expression – the data to be encrypted. Binary values can also be passed to **AES_ENCRYPT**. This parameter is case-sensitive, even in case-insensitive databases.

key – the encryption key used to encrypt the *string-expression*. To obtain the original value, you must also use the same key to decrypt the value. This parameter is case sensitive, even in case-insensitive databases.

As with most passwords, it is best to choose a key value that is difficult to guess. Sybase recommends that you choose a value for your key that is at least 16 characters long, contains a mix of uppercase and lowercase letters, and includes numbers and special characters. You need this key each time you want to decrypt the data.

Warning! Protect your key; store a copy of your key in a safe location. If you lose your key, encrypted data becomes completely inaccessible and unrecoverable.

Usage

AES_ENCRYPT returns a `VARBINARY` value, which is at most 31 bytes longer than the input *string-expression*. The value returned by this function is the ciphertext, which is not human-readable. You can use the **AES_DECRYPT** function to decrypt a *string-expression* that was encrypted with the **AES_ENCRYPT** function. To successfully decrypt a *string-expression*, use the same encryption key and algorithm used to encrypt the data. If you specify an incorrect encryption key, an error is generated.

If you are storing encrypted values in a table, the column should be of data type `VARBINARY` or `VARCHAR`, and greater than or equal to 32 bytes, so that character set conversion is not performed on the data. (Character set conversion would prevent decryption of the data.) If the length of the `VARBINARY` or `VARCHAR` column is less than 32 bytes, then the **AES_DECRYPT** function returns an error.

The result data type of an **AES_ENCRYPT** function may be a `LONG BINARY`. If you use **AES_ENCRYPT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **AES_ENCRYPT** to the correct data type and size.

For additional details and usage information, see *Reference: Building Blocks, Tables, and Procedures > SQL Functions > Alphabetical List of Functions > REPLACE function [String]*.

Standards and Compatibility

- SQL – vendor extension to ISO/ANSI SQL grammar
- Sybase – not supported by Adaptive Server Enterprise

AES_DECRYPT Function [String]

Decrypts the string using the supplied key, and returns, by default, a `VARBINARY` or `LONG BINARY`, or the original plaintext type.

Syntax

```
AES_DECRYPT( string-expression, key [, data-type ] )
```

Parameters

string-expression – the string to be decrypted. Binary values can also be passed to this function. This parameter is case sensitive, even in case-insensitive databases.

key – the encryption key required to decrypt the *string-expression*. To obtain the original value that was encrypted, the key must be the same encryption key that was used to encrypt the *string-expression*. This parameter is case-sensitive, even in case-insensitive databases.

Warning! Protect your key; store a copy of your key in a safe location. If you lose your key, the encrypted data becomes completely inaccessible and unrecoverable.

data-type – this optional parameter specifies the data type of the decrypted *string-expression* and must be the same data type as the original plaintext.

If you do not use a **CAST** statement while inserting data using the **AES_ENCRYPT** function, you can view the same data using the **AES_DECRYPT** function by passing `VARCHAR` as the *data-type*. If you do not pass *data-type* to **AES_DECRYPT**, `VARBINARY` data type is returned.

Usage

You can use the **AES_DECRYPT** function to decrypt a *string-expression* that was encrypted with the **AES_ENCRYPT** function. This function returns a `VARBINARY` or `LONG VARBINARY` value with the same number of bytes as the input string, if no data type is specified. Otherwise, the specified data type is returned.

To successfully decrypt a *string-expression*, you must use the same encryption key that was used to encrypt the data. An incorrect encryption key returns an error.

Example

Decrypt the password of a user from the `user_info` table.

```
SELECT AES_DECRYPT(user_pwd, '8U3dkA', CHAR(100))
FROM user_info;
```

Standards and Compatibility

- SQL – vendor extension to ISO/ANSI SQL grammar
- Sybase – not supported by Adaptive Server Enterprise

BFILE Function

Extracts individual `LONG BINARY` and `LONG VARCHAR` cells to individual operating system files on the server.

Syntax

```
BFILE( file-name-expression, large-object-column )
```

Parameters

file-name-expression – the name of the output file into which the `LONG BINARY` or `LONG VARCHAR` data is written. This file name can be up to (32K -1) bytes in length, but must be a valid path name that is supported by the file system.

large-object-column – the name of the `LONG BINARY` or `LONG VARCHAR` column.

Usage

BFILE returns:

- 1, if the file is successfully written
- 0, if the file is not successfully opened or written
- NULL, if the `LONG BINARY` or `LONG VARCHAR` cell value is NULL

If the `LONG BINARY` or `LONG VARCHAR` cell value is NULL, no file is opened and no data is written.

The file path is relative to where the server was started and the open and write operations execute with the permissions of the server process. Tape devices are not supported for the **BFILE** output file.

`LONG BINARY` and `LONG VARCHAR` cells retrieved other than with the **BFILE** function (that is, retrieved through the client/server database connection later) are limited in size to a maximum length of 2GB. Use **SUBSTRING64** or **BYTE_SUBSTR64** to retrieve `LONG BINARY` cells greater than 2GB using a **SELECT** (**SELECT**, **OPEN CURSOR**). Use **SUBSTRING64** to retrieve `LONG VARCHAR` cells greater than 2GB using a **SELECT** (**SELECT**, **OPEN CURSOR**). Some connection drivers, for example ODBC, JDBC™, and Open Client™, do not allow more than 2GB to be returned in one **SELECT**.

You can use **BFILE** with or without the data extraction facility.

DATE Function [Date and Time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

Syntax

```
DATE ( expression )
```

Parameters

Table 110. Parameters

Parameter	Description
expression	The value to be converted to date format. The expression is usually a string.

Returns

DATE

Example

The following statement returns the value 1988-11-26 as a date.

```
SELECT DATE( '1988-11-26 21:20:53' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

DAY Function [Date and Time]

Returns an integer from 1 to 31 corresponding to the day of the month of the date specified.

Syntax

```
DAY ( date-expression )
```

Parameters

Table 111. Parameters

Parameter	Description
date-expression	The date.

Returns

SMALLINT

Example

The following statement returns the value 12:

```
SELECT DAY( '2001-09-12' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DB_PROPERTY Function [System]

Returns the value of the given property.

Syntax

```
DB_PROPERTY ( { property-id | property-name }
[ , { database-id | database-name } ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 112. Parameters

Parameter	Description
property-id	The database property ID.
property-name	The database property name.
database-id	The database ID number, as returned by DB_ID. Typically, the database name is used.
database-name	The name of the database, as returned by DB_NAME.

Returns

VARCHAR

Example

The following statement returns the page size of the current database, in bytes.

```
SELECT DB_PROPERTY( 'PAGESIZE' ) FROM iq_dummy
```

Usage

Returns a string. The current database is used if the second argument is omitted.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

EVENT_CONDITION Function [System]

Specifies when an event handler is triggered.

To define an event and its associated handler, use the **CREATE EVENT** statement.

See *Reference: Statements and Options > SQL Statements > CREATE EVENT Statement*.

Syntax

```
EVENT_CONDITION ( condition-name )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 113. Parameters

Parameter	Definition
condition-name	The condition triggering the event. The possible values are preset in the database, and are case-insensitive. Each condition is valid only for certain event types.

Table 114. Valid conditions for events

Condition name	Units	Valid for	Comment
DBFreePercent	N/A	DBDiskSpace	DBDiskSpace shows free space in the system database file (.db file), not the IQ store.
DBFreeSpace	Megabytes	DBDiskSpace	
DBSize	Megabytes	GrowDB	
ErrorNumber	N/A	RAISERROR	
IdleTime	Seconds	ServerIdle	
Interval	Seconds	All	Time since handler last executed.
LogFreePercent	N/A	LogDiskSpace	
LogFreeSpace	Megabytes	LogDiskSpace	
LogSize	Megabytes	GrowLog	
RemainingValues	Integer	GlobalAutoincrement	The number of remaining values.
TempFreePercent	N/A	TempDiskSpace	TempDiskSpace shows free space in the system temporary file (pointed to by TEMP or IQTMP15 environment variable), not the IQ temporary store.
TempFreeSpace	Megabytes	TempDiskSpace	

Condition name	Units	Valid for	Comment
TempSize	Megabytes	GrowTemp	

Returns

INT

Example

The following event definition uses the **EVENT_CONDITION** function:

```
create event LogNotifier
type LogDiskSpace
where event_condition( 'LogFreePercent' ) < 50
handler
begin
    message 'LogNotifier message'
end
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

HOUR Function [Date and Time]

Returns a number from 0 to 23 corresponding to the hour component of the specified date/time.

Syntax

HOUR (*datetime-expression*)

Table 115. Parameters

Parameter	Definition
datetime-expression	The date/time.

Returns

SMALLINT

Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

ISDATE Function [Date and Time]

Tests whether a string argument can be converted to a date.

If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax

```
ISDATE ( string )
```

Parameters

Table 116. Parameters

Parameter	Description
string	The string to be analyzed to determine whether the string represents a valid date.

Returns

INT

Example

The following example tests whether the `birth_date` column holds valid dates, returning invalid dates as NULL, and valid dates in `date` format.

```
select birth_date from MyData;
------------------------------
1990/32/89
0101/32/89
1990/12/09
```

```
select
  case when isdate(birth_date)=0 then NULL
  else cast(birth_date as date)
  end
  from MyData;
------------------------------------
(NULL)
(NULL)
1990-12-09
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

LOCATE Function

The **LOCATE** function returns a 64-bit signed integer containing the position of the specified string in the large object column or variable parameter. For `CHAR` and `VARCHAR` columns, **LOCATE** returns a 32-bit signed integer position.

Syntax

```
LOCATE( large-object-column, string-expression
[, numeric-expression ] )
```

Parameters

large-object-column – the name of the `LONG VARCHAR` or `LONG BINARY` column or variable to search.

string-expression – the string of up to 255 bytes, for which you are searching.

numeric-expression – the character position or offset at which to begin the search in the string. The *numeric-expression* is a 64-bit signed integer for `LONG VARCHAR` and `LONG BINARY` columns and is a 32-bit signed integer for `CHAR`, `VARCHAR`, and `BINARY` columns. The first character is position 1. If the starting offset is negative, **LOCATE** returns the last matching string offset, rather than the first. A negative offset indicates how much of the end of the string to exclude from the search. The number of characters excluded is calculated as (-1 * offset) - 1.

Usage

- All the positions or offsets, returned or specified, in the **LOCATE** function are always character offsets and may be different from the byte offset for multibyte data.
- If the large object cell being searched contains more than one instance of the string:
  - If *numeric-expression* is specified, **LOCATE** starts the search at that offset in the string.
  - If *numeric-expression* is not specified, **LOCATE** returns only the position of the first instance.
- If the column does not contain the string, **LOCATE** returns zero (0).
- Searching for a string longer than 255 bytes returns NULL.
- Searching for a zero-length string returns 1.
- If any of the arguments is NULL, the result is NULL.
- **LOCATE** supports searching `LONG VARCHAR` and `LONG BINARY` columns and `LONG VARCHAR` and `LONG BINARY` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

See *Reference: Building Blocks, Tables, and Procedures > SQL Functions > Alphabetical List of Functions > LOCATE Function [String]*.

MOD Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
MOD ( dividend, divisor )
```

Parameters

Parameters	Description
dividend	The dividend, or numerator of the division.
divisor	The divisor, or denominator of the division.

Returns

SMALLINT

INT

NUMERIC

Example

The following statement returns the value 2:

```
SELECT MOD( 5, 3 ) FROM iq_dummy
```

Usage

Division involving a negative *dividend* gives a negative or zero result. The sign of the *divisor* has no effect.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise. The % operator is used as a modulo operator in Adaptive Server Enterprise.

PI Function [Numeric]

Returns the numeric value PI.

Syntax

```
PI ( * )
```

Returns

DOUBLE

Example

The following statement returns the value 3.141592653....

```
SELECT PI( * ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—The **PI()** function is supported in Adaptive Server Enterprise, but **PI(*)** is not.

ROW_NUMBER() Function

The **ROW_NUMBER** function returns a unique row number for each row.

If you define window partitions, **ROW_NUMBER** starts the row numbering in each partition at 1, and increments each row by 1. If you do not specify a window partition, **ROW_NUMBER** numbers the complete result set from 1 to the total cardinality of the table.

The **ROW_NUMBER** function syntax is:

```
ROW_NUMBER() OVER ([PARTITION BY window partition] ORDER BY window
ordering)
```

ROW_NUMBER does not require an argument, but you must specify the parentheses.

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

BIT_LENGTH Function [String]

Returns an unsigned 64-bit value containing the bit length of the column parameter.

Syntax

```
BIT_LENGTH( column-name )
```

Parameters

Table 117. Parameters

Parameter	Description
column-name	The name of a column

Returns

INT

Usage

The return value of a NULL argument is NULL.

The **BIT_LENGTH** function supports all Sybase IQ data types.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Unstructured Data Analytics in Sybase IQ > Function Support*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by SQL Anywhere or Adaptive Server Enterprise.

Supported Sybase IQ Functions for SQL Transformation Project

To review the list of Sybase IQ functions that can be used in a SQL Transformation project, see *Sybase IQ Reference: Building Blocks, Tables, and Procedures*.

Connection Parameters

Learn about the database configuration options and review additional information for some of the supported interfaces.

Interface-Specific Database Options

The interface-specific database options table provides database options available for each of the interfaces in an ETL project.

In the table:

- A hyphen (-) indicates that the database option has no default value. You can enter an appropriate value.
- "NA" indicates that the database option is not provided or available for the underlying interface.
- "Not used" indicates that the database options is not used by the underlying interface, though it is displayed.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persistent	
Always use logon credentials	NA	0	NA	NA	NA	NA	When building the ODBC connection string, always add the credentials to the connection string.
API trace	NA	NA	False	False	NA	NA	Enable CTLIB trace facility.
API version	NA	NA	150	125	NA	NA	CTLIB API version compatibility.
Auto vacuum	NA	NA	NA	NA	NA	0	Reclaims the space when objects are deleted from the database.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persistent	
BLOB chunk size	1024 While fetching LOB data from the file, Sybase IQ InfoPrimer fetches 1024 bytes from the file and writes them to the database each time.	NA	NA	NA	NA	NA	Determines the size at which LOBs are truncated.
BLOB fetch mode	LOB_ INLINE	INLINE	NA	NA	NA	NA	BLOB data is written either to the secondary file or held in the memory. If set to INLINE, data is held in memory. If set to FILE, data is written temporarily to the disk.
Busy timeout	NA	NA	NA	NA	NA	10	Creates a handler, which waits for the specified number of seconds on encountering a locked database table.
Cache size	NA	NA	NA	NA	NA	3000	Number of pages to use in cache.
CLIENT_ CHARSET	NA	NA	-	-	NA	NA	User defined character set to use with Client Library (CTLIB).
CLIENT_ CONVERSION	NA	NA	0 (ASE) 1 (ASA/IQ)	0 (ASE) 1 (ASA/ IQ)	NA	NA	Controls whether or not the client library should convert data to the appropriate form.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persis- tent	
Connect timeout	0 (Not used)	0 (Not used)	0	0	10	0	Stops trying to con- nect after the number of Connect timeout seconds. If set to 0, the connect does not timeout.
CONVERT- ER_CHAR- SET	NA	NA	-	-	NA	NA	The character set to be used when CLI- ENT_ CONVERSION = 1.
Database name	NA	NA	-	-	NA	NA	Database name.
DBMS_VE R	NA	NA	-	-	NA	NA	Database version.
Default cache size	NA	NA	NA	NA	NA	3000	Default number of pages to use in cache.
Disconnect timeout	NA	10 On Win- dows 32- bit, Syb- ase IQ Info- Primer always uses the default value.On other plat- forms, this op- tion is not used.	NA	NA	NA	NA	Enforces disconnec- tion from the data- base, if there is no re- ply from the database for *n* seconds after you try to disconnect.
Enable SQL Server fast load	NA	NA	NA	NA	1	NA	If set to 1, the MS SQL Server fast load feature is enabled. If set to 0, the feature is disabled.
Execution timeout	0 (Not used)	0 /Not used	-1	-1	NA	0	Component stops ex- ecution after a time interval in seconds. (0 <= means no time- out).

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persis- tent	
Extended connect op- tions	NA	-	NA	NA	-	NA	Allows additional driver-specific pa- rameters to be added to an ODBC connec- tion string.
Full column names	NA	NA	NA	NA	NA	1	When set to 1, col- umn names are fully qualified, following this pattern: <table- name/alias> <col- umn-name>.
Internal data- base	NA	NA	NA	NA	NA	-	Database reference.
Isolation lev- el	DE- FAULT (Not used)	DE- FAULT	DEFAULT (Not used)	DE- FAULT (Not used)	NA	DE- FAULT (Not used)	Defines the degree to which one transac- tion must be isolated from resource or data modifications made by other transactions.
Lock result- set data	0 (Not used)	0	0 (Not used)	0 (Not used)	NA	0	Query tables will be locked ensuring that no data is written to the selected record set while the process is working on it. The se- lected record set is re- leased when the last record from that set is fetched.
Log SQL statements to a file	0	0	0	0	0	0	If set to 1, all SQL statements are logged to the log or `SQL.log` file.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persis- tent	
Numeric support	NA	0 This val- ue is 0 rather than the user in- put, when DBMS is IQ or ODBC driver is ASA9.	NA	NA	NA	NA	Whether or not to en- able ODBC numeric support.
Object name end quote	"	- ODBC uses the value queried from DBMS rather than the user in- put.	]	]	-	NA	When creating SQL statements, terminat- ing character are used as quotes.
Object name start quote	"	"" ODBC uses the value queried from DBMS rather than the user in- put.	[	[	-	NA	Beginning character to be used as quote when building SQL statements.
PAD_BLAN KS	NA	NA	0	0	NA	NA	Maintains a constant column width using space characters.
Page size	NA	NA	NA	NA	NA	4096	Number of bytes per page. Must be a pow- er of 2, greater than or equal to 512, and not higher than 32768.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persis- tent	
Quote char- acter	"	NA	NA	NA	NA	NA	Same as QUOTE_START and QUOTE_END.
Quote object names	0 (Not used)	0	0	0	0	0	If set to 1, the charac- ter specified in QUOTE_START and QUOTE_END is used to surround identifiers in gener- ated SQL statements.
Reject log column de- limiter	tab	tab	tab	tab	-	tab	Used as a column de- limiter in the reject log.
Short col- umn names	NA	NA	NA	NA	NA	0	If flag is set to false, column names are fully qualified, other- wise they are referred to only by the column name.
Show all ta- bles	NA	NA	0 (Not used)	0 (Not used)	NA	NA	Display system tables as well as user tables.
Show error location	1	1	1	1	1	1	Display the error lo- cation if set to 1. Da- tabase errors include the position of the re- cord within the result set.
SHOW_ ER- ROR_ LO- CATION_ ABSO- LUTE_ ROWS	1	1	1	1	1	1	Show the error loca- tion from the absolute begining of the result set, rather than the current result set.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persis- tent	
Synchronous	NA	NA	NA	NA	NA	0	If you select Full =2, SQLite ensures data is written to disk before continuing. If you select Normal=1, SQLite pauses to write at critical moments but not as frequently as when set to 2. If you select Off = 0, data is handed off to operating system and SQLite continues.
Temp store	NA	NA	NA	NA	NA	2	If set to 1, the location of the temporary database is a file. If set to 2, the location of the temporary database is memory.
Treat numeric value as character	NA	1 If database is IQ, or driver name contains “SYSYBNT” or “LIBDB2.A,”ODBC uses 1 instead of the user input.	NA	NA	NA	NA	Force conversion of numeric data to string.
Use system views	True	NA	NA	NA	NA	NA	Use DBA system tables to show metadata instead of per-user metadata.

DB Option	Interface and Default Value						Description
	Oracle	ODBC/ DB2	Sybase 15	Sybase	OLEDB	SQLite Persistent	
Validate result column binding	NA	NA	NA	NA	1	NA	If set to 1, the result column mapping binding is validated when reading data from the database.
Write empty dates as NULL	0	0	0 (Not used)	0 (Not used)	NA	NA	If set to 1, the value of empty dates are enforced to be NULL.
Write rejected records to file	-	-	-	-	-	-	Specifies the file path for reject logs. This option is used to log records that are rejected by the database on loading.

Database and Interface Support

The Database and Interface Support Matrix table lists the interfaces and databases available for each database component type in an ETL or EL project .

In the table, NS indicates the interface is not supported for the database component or the EL project.

Interface	DB Data Provider and DB Lookup	DB Data Sink	DB Bulk Load Sybase IQ	DB Staging	IQ Loader File via Load Table	IQ Loader DB via Insert Location	SQL Executor	EL Project
Sybase	ASE IQ ASA	IQ	IQ	ASE ASA IQ	IQ	IQ	ASE IQ ASA	ASE IQ ASA
ODBC	Any datasource that can be accessed via ODBC.	IQ	IQ	IQ ASA ASE	IQ	NS	ASE IQ SQL Server ASA	Any datasource that can be accessed via ODBC. Only supports an ODBC connection to IQ as a destination.
OLE DB	SQL Server	NS	NS	NS	NS	NS	SQL Server	SQL Server

Interface	DB Data Provider and DB Lookup	DB Data Sink	DB Bulk Load Sybase IQ	DB Staging	IQ Loader File via Load Table	IQ Loader DB via Insert Location	SQL Executor	EL Project
Oracle	Any Oracle database system that can be accessed by Oracle Call Interface (OCI).	NS	NS	NS	NS	NS	Oracle	Any Oracle database system that can be accessed by Oracle Call Interface (OCI).
DB2	Any DB2 database system that can be accessed by the IBM DB/2 client interface.	NS	NS	NS	NS	NS	DB2	Any DB2 database system that can be accessed by the IBM DB/2 client interface.
Insert Location	NS	NS	NS	NS	NS	NS This component uses the Sybase Interface for **INSERT LOCATION**.	NS	Any remote server defined on the destination IQ server.

The Sybase IQ InfoPrimer environment has been tested, evaluated, and verified thoroughly to comply with many interface drivers of the supported database systems. If you encounter unexpected results that might be related to driver incompatibility, try installing one of the supported versions for your interfacing driver. See Interface support in the *Sybase IQ InfoPrimer Release Bulletin* for a list of all supported interface driver versions.

Working with the SQLite Persistent Interface

Sybase IQ InfoPrimer technology includes a built-in, general purpose, relational database you can use for temporary data storage and staging. It is based on SQLite, a very fast, widely used, mostly SQL92-compliant database. SQLite is a small C library that implements a self-contained, embeddable, zero configuration SQL database engine.

Connecting to a SQLite Database

A SQLite database is represented as a single file with the `.db` extension. The database file can contain any number of tables.

1. In the Properties window, select **SQLite Persistent** from the **Interface** menu.
2. Provide the host name for the SQLite database file.

- To create a new SQLite database file, provide a name in the **Host Name** field; do not include the `.db` extension. A new SQLite database file, with the extension `.db`, is automatically created in the default location, which is the `database` directory under the installation folder.

  To create the SQLite database file in a directory other than the default, specify the complete path, including the `.db` extension, in the **Host Name** field.
- If you are connecting to an existing SQLite database file in the default directory, select the file name from the **Host Name** menu. Do not enter the `.db` extension. To connect to a SQLite database file in a directory other than the default, specify the complete path, including the `.db` extension, in the **Host Name** field.

For example, to create a new SQLite database file called `mySQLite.db`, or to connect to an existing `mySQLite.db` database file, use these parameters:

- **Interface**: SQLite Persistent
- **Host Name**: mySQLite

Creating a SQLite Table

Create a SQLite table using one of the specified ways.

- Right-click a Staging component and select **Create Staging Table from Input** or **Create Staging Table from Port**.
- Right-click one of the Data Sink components and select **Add Destination Table from Input** or **Add Destination Table from Port**.

Extracting Data from a SQLite Database

Provide the proper connection parameters for the SQLite database file on a DB component.

You can use SQLite-supported SQL commands in the preprocessing or postprocessing SQL properties of components connected to databases.

Use the Content Explorer from the **Tools** menu to manipulate or browse objects of the SQLite database connected to components in your project. You can also use client applications available from *sqlite.org* to connect to SQLite database files.

Note: To use external client applications to connect to your SQLite database files, you must be familiar with the locking strategy of SQLite.

Working with the Oracle Interface

Review the class-level translations for Sybase IQ InfoPrimer datatypes to Oracle datatypes for an EL or ETL project.

Table 118. Data Type Mapping from Oracle Interface to Sybase IQ InfoPrimer

Sybase IQ Info-Primer Datatype	Oracle Datatype	Size/Precision	Minimum Scale	Maximum scale
binary	BLOB	2147483647		
binary	BFILE	2147483647		
binary	RAW	2000	0	0
string	CLOB	2147483647	0	0
string	CHAR	2000	0	0
float	DECIMAL	38	0	0
integer	NUMBER	38	0	0
float	DOUBLE PRECI-SION	15	0	38
datetime	DATE	19	0	0
datetime	TIMESTAMP	28	0	9
string	VARCHAR2	4000	0	0
unicode	NCHAR	1000	0	0
unicode	NVARCHAR2	2000	0	0
unicode	NCLOB	2147483647	0	0

Best Practices

Review the best practices for Sybase IQ InfoPrimer.

Best Practices for Working with Sybase IQ InfoPrimer Server

Review the best practices for working with Sybase IQ InfoPrimer Server.

Do Not Start Multiple Sybase IQ InfoPrimer Server Sessions

If you start Sybase IQ InfoPrimer Server from the command line while Sybase IQ InfoPrimer Development is running, Sybase IQ InfoPrimer Development becomes unstable and displays error messages if you perform any action in Sybase IQ InfoPrimer Development. This is due to conflicts between the Sybase IQ InfoPrimer Server session you started from the command line and the Sybase IQ InfoPrimer Server session started by Sybase IQ InfoPrimer Development.

To avoid starting multiple server sessions, in the Preferences window, unselect **Engine > Start local engine during application start-up**. This prevents the Sybase IQ InfoPrimer Server session from starting automatically next time you start Sybase IQ InfoPrimer Development.

Enter the Default Port Number for Command Line Execution

Command line execution fails if you attempt to use the default port number of 5124 but you do not include it in the command line.

You must enter the default port number of 5124 in the command line, for example:

```
GridNode -con --port 5124 --server localhost
-f "..\\testdata\\tpms\\tpms_TestB.xml"
```

Use Column Aliases When Entering Queries

Output column names for query result sets are generated by the source database. When you apply a function to an attribute in a query, the output column name is different for different databases and may bear no relation to the source column name. You can define custom column names to be used as port attribute names and shown in the Data Viewer, by adding column aliases when you enter queries.

For example, when querying an Adaptive Server Enterprise database using an aggregate function, the Content Browser displays the results of the query without a column header for the aggregated data column. To add a column header, add an alias value on the same row as the attribute, which utilizes the aggregate function. If the query is:

SELECT COUNT (qsID) FROM TAB_IDS, add an alias value to the returned column:

SELECT COUNT (qsID) AS qsID_COUNT FROM TAB_IDS

Avoid System Error in JavaScript Editor While Debugging Large Scripts

The default maximum size of the JavaScript engine runtime is 20000000 bytes, or 19.07MB. Debugging large scripts or long running scripts such as a script with many loop iterations, you may encounter a memory issue.

To avoid this issue, configure the maximum memory size for the JavaScript engine in the `Default.ini` file.

- Navigate to the `etc` directory of the installation folder and use a text editor to open the `Default.ini` file.
- In the [Scripting] section, add:

  ```
  Runtime Memory = <a new number for memory in bytes>
  ```

Do Not Perform DDL Operations in Transactional Projects

Database target or interface combinations behave differently for data definition language (DDL) transactions, so avoid performing DDL operations in a preprocessing sql statement and postprocessing sql statement for transactional projects. If needed, perform DDL operations in a nontransactional project.

For example, create a job that has three projects:

- Non-transactional setup project (including DDL)
- Transactional project
- Non-transactional cleanup project (including DDL)

Best Practices for Working with ETL Project Components

Review the best practices for working with ETL project components.

Migrating Wide Tables

Migrating tables with hundreds or thousands of columns consumes a lot of memory. To prevent errors while migrating wide tables with varying numbers of columns and rows from a source Sybase IQ database to a target Sybase IQ database, follow the recommendations described in this section.

- Allocate a 1GB database space for the SQL Anywhere repository with all the other values remaining as system defaults.
- Use the following techniques, components, and interfaces to migrate tables with large number of columns:

Number of Columns	Migration Techniques	Interface to Use
Up to 3000 columns	Insert Location component	Sybase
Up to 3000 columns	Load Table component	Sybase or ODBC
Up to 3500 columns	Migration wizard **Note:** The Migration wizard may fail to generate the migration projects, either due to memory limitations, or if you do not use a SQL Anywhere repository.	Sybase or ODBC
Up to 10000 columns	Load Table component	ODBC

Note: When you are migrating a table with more than 3000 columns, you may encounter performance issues moving large number of rows.

Importing an XML File with More than 32 Sibling Elements

To import more than 32 sibling elements using the XML via SQL Data Provider component, set the Create Flat View to 0 in the XML Options property of the component. You must manually set up sub queries using the Content Explorer.

Loading Last Row of Source Text File to Sybase IQ

When you are using the IQ Loader File via Load Table component, Sybase IQ does not accept the last row of a source text file from Sybase IQ InfoPrimer if the last row does not end with a trailing row delimiter.

To avoid this issue, add a line delimiter at the end of the last row in the source text file. For example, in Windows, with the row delimiter specified as CRLF, place the cursor at the end of the last row and press Enter to add a row delimiter to the last row.

Configuring Adaptive Server Enterprise for Bulk Copying

If you are using Adaptive Server Enterprise for DB Staging, you must first configure the Adaptive Server database for bulk copying. If the Adaptive Server is not configured, you may encounter errors, although the project executes successfully.

Data Calculator JavaScript and DB Staging Components Limitation

Do not add more than 35 Data Calculator JavaScript and DB Staging components to a single project.

Increasing the Text Size for the Adaptive Server ODBC Driver

The Adaptive Server ODBC driver truncates `text` or `image` data values that are larger than the value set in the ODBC configuration in Microsoft Windows for the driver. You must increase the text size value in the ODBC Control Panel, or set the value in the database options parameter for your database connection to avoid this issue.

Increasing the Text Size Value Using the Control Panel

Use the Control Panel to update the text size value.

1. Select **Control Panel > Administrative Tools > Data Sources (ODBC)**, then select the datasource for Adaptive Server Enterprise in the **User DSN** or **System DSN** tab.
2. Select **Configure** to display the ODBC Adaptive Server Enterprise Setup window, then select **Advanced.**
3. Change the value for Text Size to a value larger than 32KB, which is the default. The Adaptive Server ODBC drive truncates any data value that is larger than the value you set here.

Increasing the Text Size Value Using Database Options

Use the Properties window for the database connection to set the text size value.

1. Double-click the edit icon of the **Database Options** field to display the Enter Properties window.
2. In the **Extended Connect Options** field, enter “TEXTSIZE=*N*” where *N* is the text size value you want to set.

Do Not Change Delimiters in the Source Text File When Project Is Executed on Different Platforms

When you create a project using the Text Data Provider component on Windows and then execute it on UNIX or Linux, make sure the source text file is not converted to the UNIX or Linux format. The delimiters used in the source text file should always be same as what was designed in the Text Data Provider component.

If the delimiters are inconsistent, you will encounter execution errors.

Setting Named Pipe Permission on Windows

For the DB Bulk Load Sybase IQ component, if you want to provide a pipe name in the Load Stage property field, you must first make some settings to avoid any errors.

1. Go to **Control Panel > Administrative Tools > Local Security Policy > Local Policies > Security Options**.
2. Double-click **Network access: Named Pipes that can be accessed anonymously** from the Policy list.
3. Add your named pipe to the existing list. For example, if the pipe name is "pipe://mypipe", add "mypipe" to the list. Click **Apply.**
4. Click **OK.**

Migrating Tables to IQ Containing LOB Columns

Before migrating tables to IQ that contain character large object (CLOB), binary large object (BLOB), image or text columns with the IQ Loader DB via Insert Location component, review your IQ configuration.

The IQ parameter LOAD_MEMORY_DB controls how much system memory IQ uses to manage these types of columns. By default, IQ sets this parameter to 0, which means that the memory usage for processing these requests is unlimited. You may see this error:

```
ASA Error -1006042: All available virtual memory
has been used; allocation cancelled:
Extra info: 948472704].
```

If you set this parameter to a specific value, for example, 300 (MB), IQ uses only the specified amount of memory to process these requests, and prevents the error from occurring. For more information on IQ parameters, see *Sybase IQ 12.7 Reference Manual* > Database Options.

Simulation Does Not Roll Back Data When a PostProcessing SQL Statement Error Occurs

Do not select the “Execute post-processing as for successful execution” option, in case a postprocessing SQL statement error occurs during execution.

If you do so, the project commits all transactional components, regardless of the error.

Best Practices for Working with Internationalization

Review the best practices for working with internationalization.

Parsing Source Files with Byte-Order Mark Correctly

If you are using the Fixed by Bytes property to parse your file, make sure the source file does not include the byte-order mark.

Use a text editor to remove the byte-order mark from the source file before parsing it.

Set Sybase IQ InfoPrimer to Support UTF-8 Encoding

Arguments given to the **uSetEnv** function cannot contain multibyte or non-Western characters.

You must set Sybase IQ InfoPrimer to support UTF-8.

For example:

```
set LANG=zh.UTF-8
```

Select Correct Encoding to Display Unicode Characters

Characters display incorrectly unless you select the correct “endianness” type for the character set encoding for Unicode files that have a byte order marker (BOM), when you load character data using the “Text Data Provider” or “Text Data Sink” components.

To display the Unicode characters correctly, in the Character Encoding field of the component configuration window, select the character set encoding with the correct endianness type for character data. For example, select:

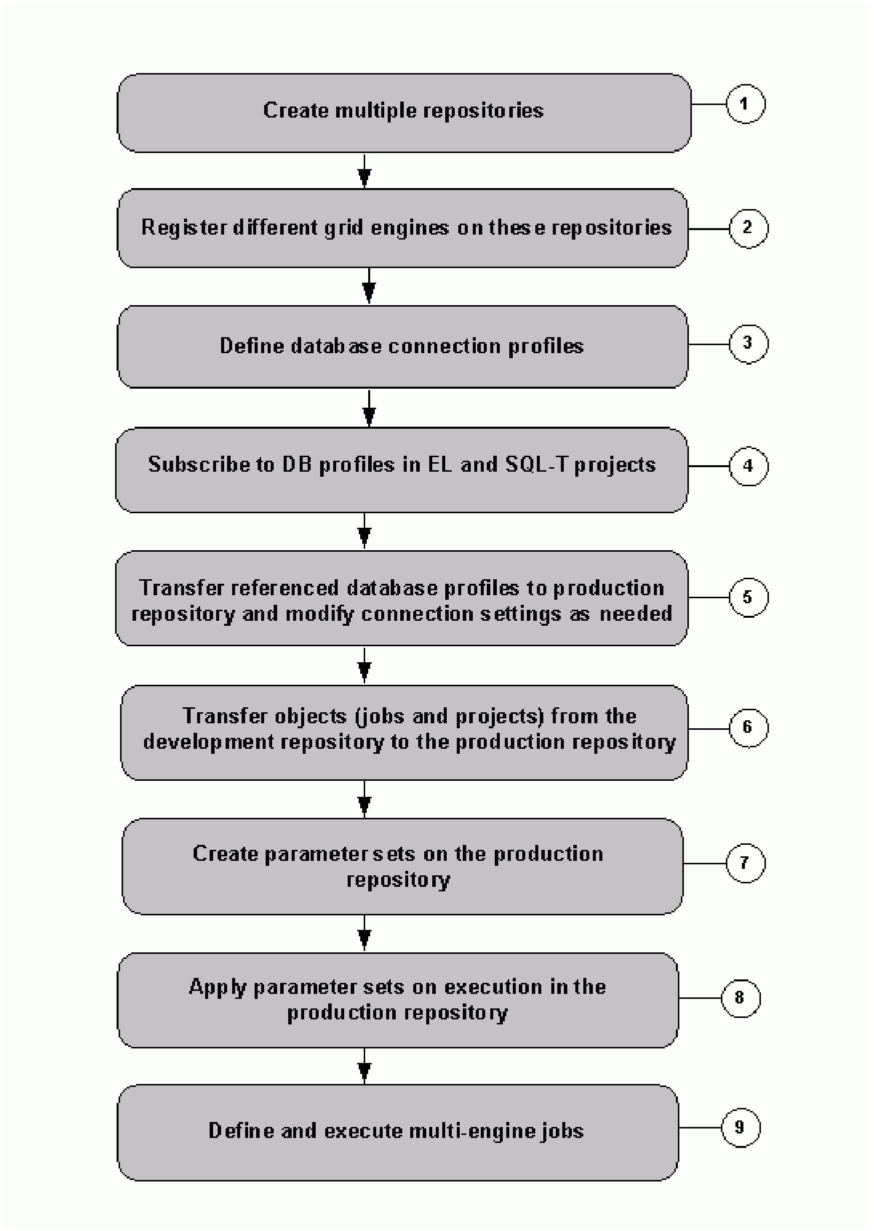
- UTF-16LE – to process text files encoded in UTF-16LE that have a BOM at the beginning of the file where LE means “little-endian” since the BOM is at the beginning of the file.
- UTF-16BE – to process text files encoded in UTF-16BE with a BOM at the end of the file where BE means “big-endian” since the BOM is at the end of the file.

Sybase IQ InfoPrimer Use Cases

Review the use cases for the various tasks you may need to perform using the Sybase IQ InfoPrimer tool. Each use case includes a graphical and sequential representation of the tasks to be performed, along with callouts. These callouts identify and provide references to the sections that include detailed information on how to perform each of the tasks.

Separating Development from a Production Environment

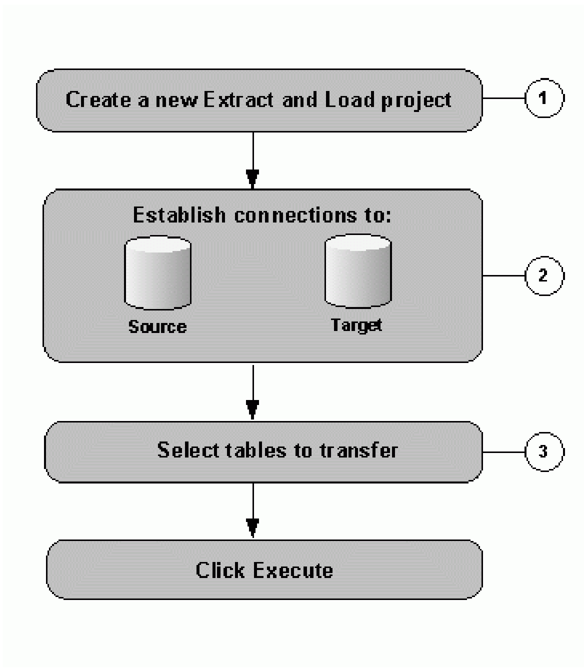
Goal: Separate a development environment from a production environment.


Create multiple repositories
1
Register different grid engines on these repositories
2
Define database connection profiles
3
Subscribe to DB profiles in EL and SQL-T projects
4
Transfer referenced database profiles to production repository and modify connection settings as needed
5
Transfer objects (jobs and projects) from the development repository to the production repository
6
Create parameter sets on the production repository
7
Apply parameter sets on execution in the production repository
8
Define and execute multi-engine jobs
9

Callout	Procedures and References
1	*Adding a Repository* on page 14 *Creating a Client and a Client User* on page 15
2	*Grid Engine Registeration* on page 23
3	*Managing Database Connection Profiles* on page 20
4	*Managing Database Connection Profiles in EL and SQL Transformation Projects* on page 30
5	*Working with Profiles on Multiple Repositories* on page 20
6	*Transferring Projects and Jobs* on page 16
7	*Parameter Sets* on page 209 • *Selecting Component Properties as Execution Parameters* on page 210 • *Managing Parameter Sets* on page 210
8	*Assigning Parameter Values* on page 211 • *Assigning the Same Value to Multiple Properties* on page 212 • *Sorting the Parameter List* on page 212 • *Special Property Values* on page 212
9	*Reducing Job Execution Time Using Multiple Engines* on page 209

Migrating Data to Sybase IQ

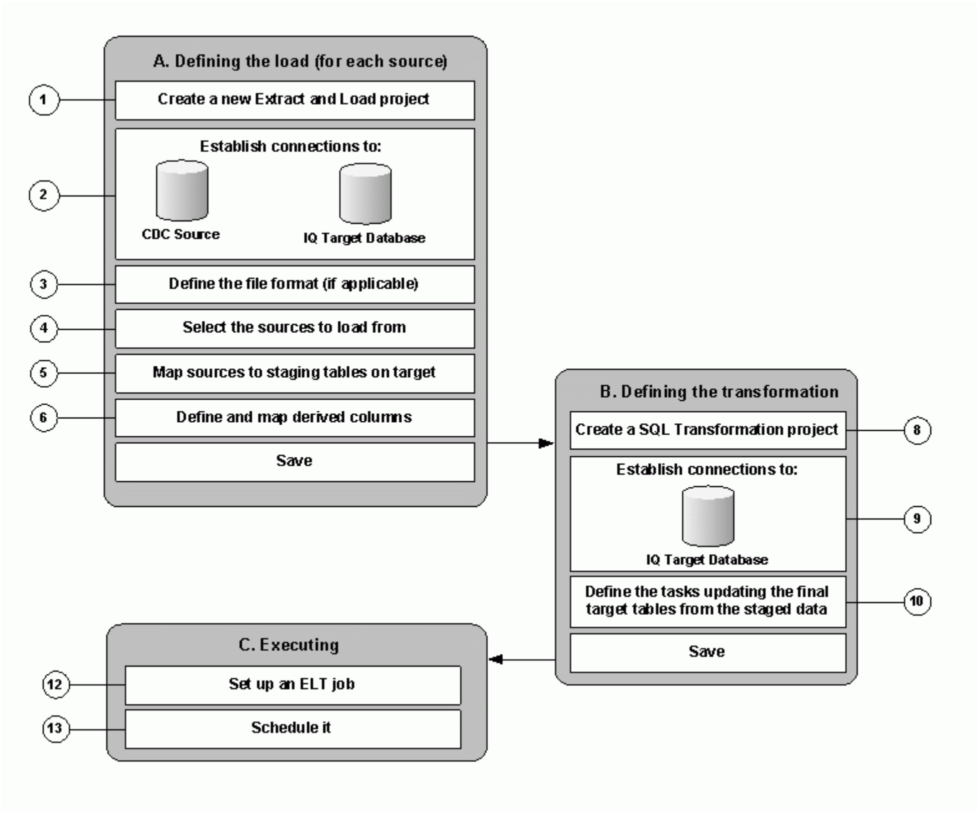
Goal: Migrate data from another database instance to Sybase IQ, if database schema is the same. Migrate data from Oracle, Adaptive Server Enterprise (ASE), Microsoft SQL Server, flat files, and other supported datasources to a Sybase IQ database, using an EL project.


Create a new Extract and Load project
1
Establish connections to:
Source
Target
2
Select tables to transfer
3
Click Execute

Callout	Procedure and Reference
1	*Configuring an EL Project* on page 33
2	• *Defining a Datasource* on page 33 • *Defining a Data Destination* on page 36
3	*Reviewing the Source and Destination Table Details* on page 36

Performing Incremental Loads

Goal: Update data in the IQ target with changed data from the source. This assumes changed data is captured in a database or data files.

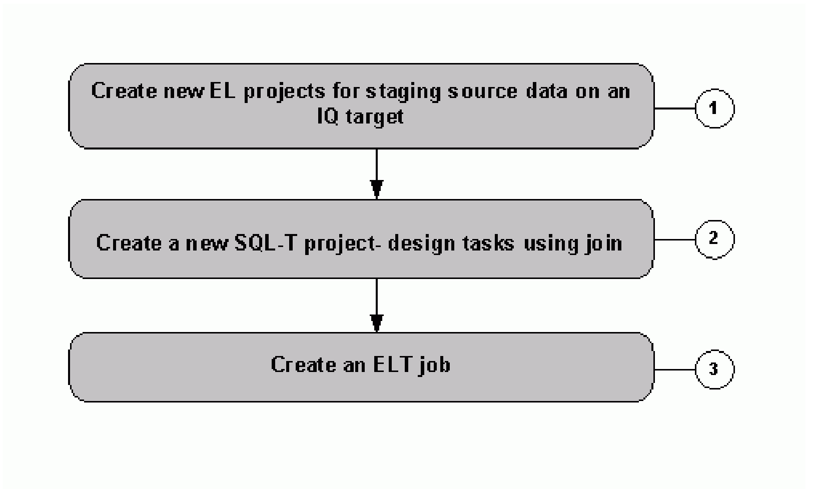

A. Defining the load (for each source)
Create a new Extract and Load project
Establish connections to:
CDC Source
IQ Target Database
Define the file format (if applicable)
Select the sources to load from
Map sources to staging tables on target
Define and map derived columns
Save
B. Defining the transformation
Create a SQL Transformation project
Establish connections to:
IQ Target Database
Define the tasks updating the final target tables from the staged data
Save
C. Executing
Set up an ELT job
Schedule it
1
2
3
4
5
6
8
9
10
12
13

Callout	See
1	*Configuring an EL Project* on page 33
2	*Defining a Datasource* on page 33and *Defining a Data Destination* on page 36
3	*Configuring a File Format* on page 44
4	*Reviewing the Source and Destination Table Details* on page 36
5	
6	*Specifying Advanced Options* on page 42

Callout	See
8	*Creating Sample SQL Transformation Projects* on page 56
9	
10	
12	*Creating a Job* on page 195
13	*Scheduling a Project And Job* on page 214

Merging Data from Different Sources

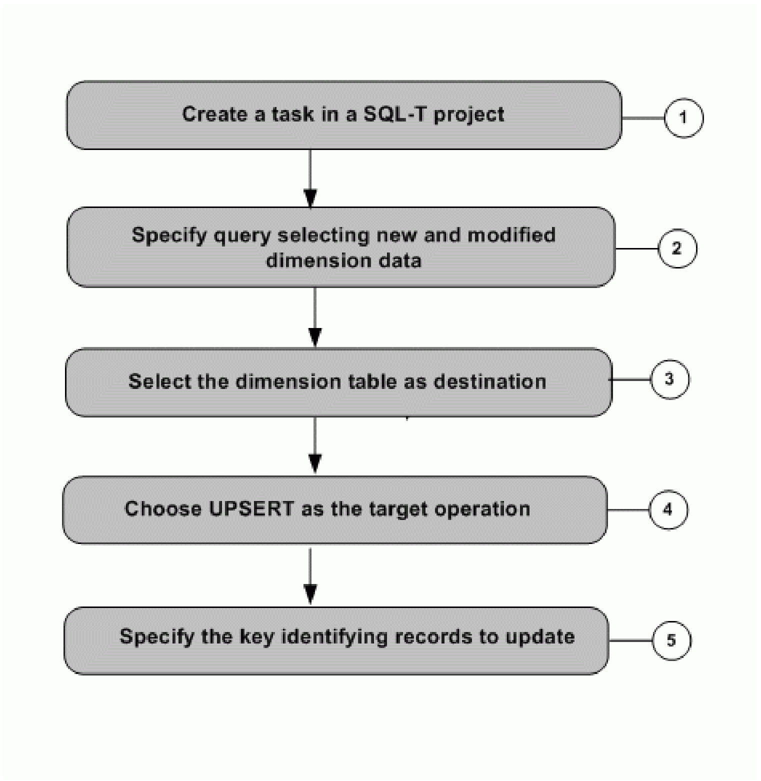
Goal: Populate target tables with joined data from multiple heterogeneous sources.


Create new EL projects for staging source data on an IQ target
1
Create a new SQL-T project- design tasks using join
2
Create an ELT job
3

Callout	See
1	*Configuring an EL Project* on page 33
2	*Creating Sample SQL Transformation Projects* on page 56
3	*Creating a Job* on page 195

Loading Dimensions Reflecting the Present

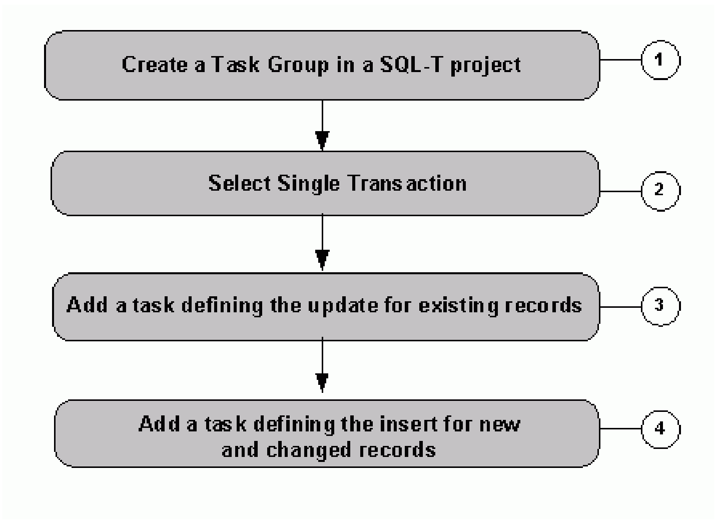
Goal: Load dimension tables. Maintaining data in a dimension table typically requires inserting new and updating changed records (slowly changing dimensions type 1 and higher). This use case assumes that changed data is already staged in IQ.


Create a task in a SQL-T project
1
Specify query selecting new and modified dimension data
2
Select the dimension table as destination
3
Choose UPSERT as the target operation
4
Specify the key identifying records to update
5

Callout	See
1	*Performing Insert Operations Using a SQL Transformation Project* on page 58 *Performing Update and Upsert Operations Using a SQL Transformation Project* on page 60
2	
3	
4	
5	

Loading Dimensions Tracking History

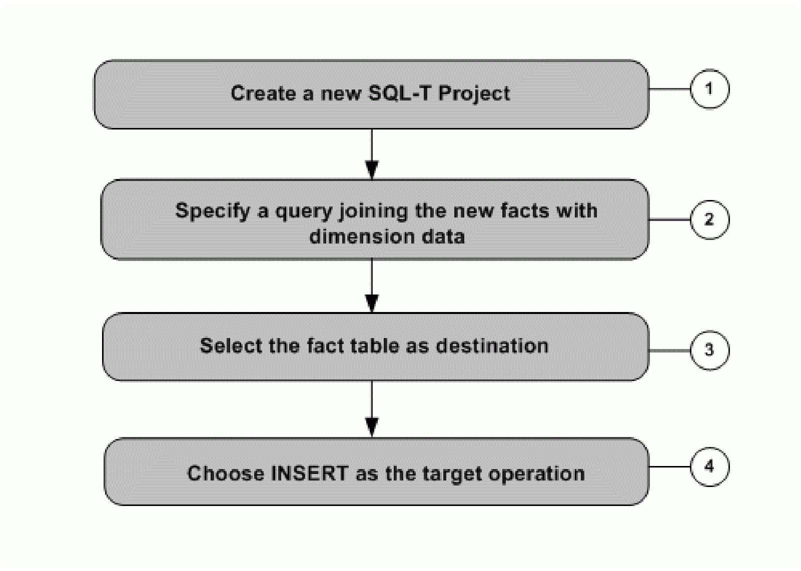
Goal: Track history of data. A special type of dimension table tracks historic data (slowly changing dimensions type 2 and higher). For existing records, these tables expect an update and an insert.


Create a Task Group in a SQL-T project
1
Select Single Transaction
2
Add a task defining the update for existing records
3
Add a task defining the insert for new and changed records
4

Callout	See
1	*Task Group* on page 51
2	
3	*Performing Update and Upsert Operations Using a SQL Transformation Project* on page 60
4	*Performing Insert Operations Using a SQL Transformation Project* on page 58

Loading Facts

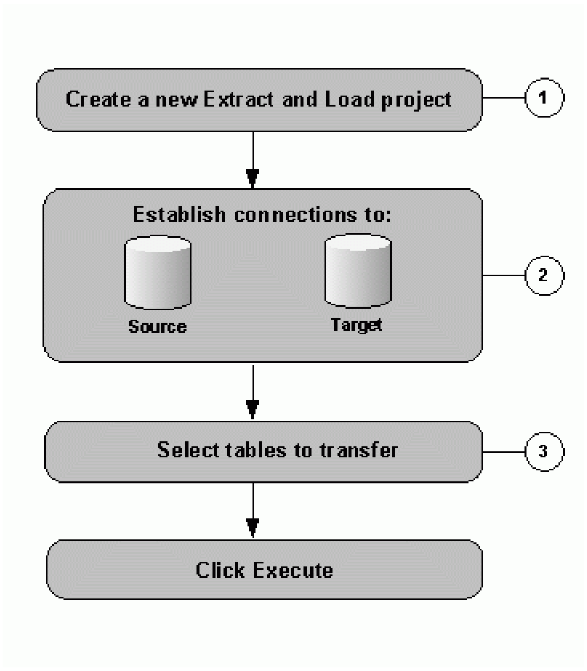
Goal: Maintaining data in a fact table typically requires inserting new data and looking up keys from dimension tables. This use case assumes that new data is already staged in IQ and dimension tables have already been updated.


Create a new SQL-T Project
1
Specify a query joining the new facts with dimension data
2
Select the fact table as destination
3
Choose INSERT as the target operation
4

Callout	See
1	*Performing Insert Operations Using a SQL Transformation Project* on page 58
2	
3	
4	

Setting up an ELT Job

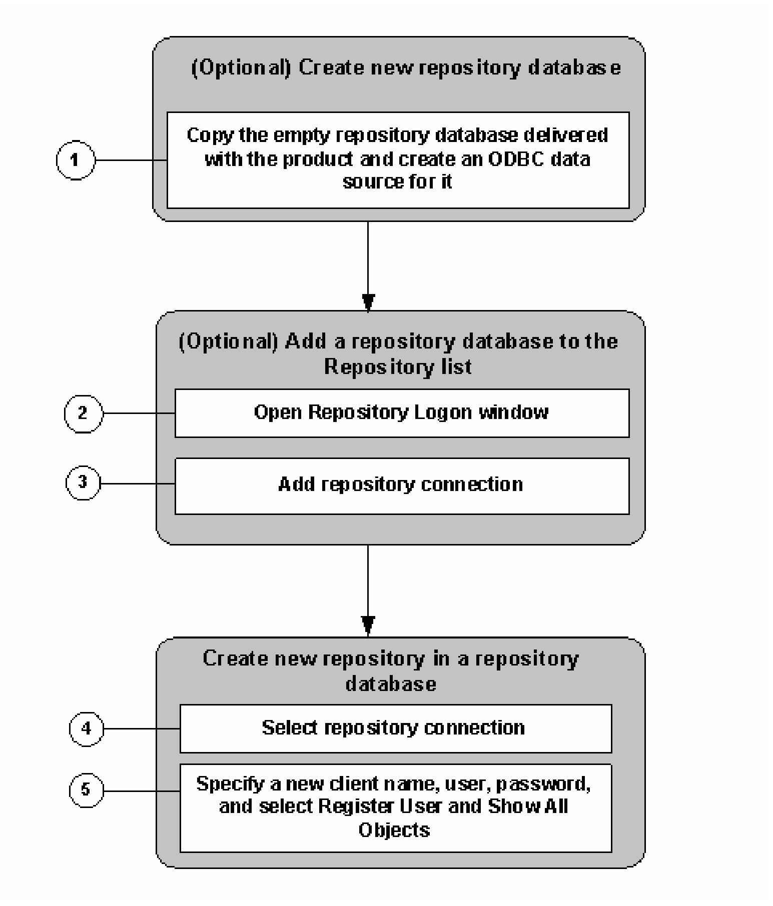
Goal: Create a job that performs all necessary steps for loading the final target.


Create a new Extract and Load project
1
Establish connections to:
Source
Target
2
Select tables to transfer
3
Click Execute

Callout	See
1	*Creating a Job* on page 195
2	*Project* on page 197 and *Multi-Project* on page 199
3	*Multi-Project* on page 199 (success port) or *Synchronizer* on page 198

Creating a Repository

Goal: Access and define the ODBC datasource for an empty SQL Anywhere 11 repository.


(Optional) Create new repository database
Copy the empty repository database delivered with the product and create an ODBC data source for it
1
(Optional) Add a repository database to the Repository list
2
Open Repository Logon window
3
Add repository connection
Create new repository in a repository database
4
Select repository connection
5
Specify a new client name, user, password, and select Register User and Show All Objects

Callout	See
1	*Configuring a Repository Database* on page 13
2	*Opening a Repository Connection* on page 14
3	*Adding a Repository Connection* on page 14
4	*Opening a Repository Connection* on page 14
5	*Creating a Client* on page 15

Index