Using the InterSystems SQL Gateway

Version 2019.2
2019-06-13

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using the InterSystems SQL Gateway
InterSystems IRIS Data Platform Version 2019.2 2019-06-13

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

About This Book

The InterSystems SQL Gateway provides access from InterSystems IRIS Data Platform™ to external databases via JDBC and ODBC. You can use various wizards to create links to tables, views, or stored procedures in external sources, allowing you to access the data in the same way you access any InterSystems IRIS object:

- Access data stored in third-party relational databases within InterSystems IRIS applications using objects and/or SQL queries.
- Store persistent InterSystems IRIS objects in external relational databases.
- Create class methods that perform the same actions as corresponding external stored procedures.

This book covers the following topics:

- Using the SQL Gateway — gives an overview of the Gateway and describes how to link to external sources.
- Connecting with the JDBC Driver — describes how to create a JDBC logical connection definition for the SQL Gateway.
- Connecting with the ODBC Driver — describes how to create an ODBC logical connection definition for the SQL Gateway.
- Using the ODBC SQL Gateway Programmatically — describes how to use the %SQLGatewayConnection class to call ODBC functions from ObjectScript.

Related Documents

The following documents contain related material:

- *Using InterSystems SQL* — describes how to use InterSystems SQL, which provides standard relational access to data stored in an InterSystems IRIS database.
- *Using Java with the InterSystems JDBC Driver* — describes how to connect to InterSystems IRIS from an external application using the InterSystems JDBC driver, and how to access external JDBC data sources from InterSystems IRIS via SQL.
- *Using the InterSystems ODBC Driver* — describes how to connect to InterSystems IRIS from an external application via InterSystems ODBC, and how to access external ODBC data sources from InterSystems IRIS.

1

Using the InterSystems SQL Gateway

The InterSystems SQL Gateway provides access from InterSystems IRIS Data Platform™ to external databases via JDBC and ODBC. This chapter discusses the following topics:

- Architecture of the InterSystems SQL Gateway — describes the internal structure and constraints of the SQL Gateway
- Creating Gateway Connections for External Sources — gives an overview of *logical connection definitions*, which are used by the SQL Gateway wizards to identify the external databases.
- The Link Table Wizard: Linking to a Table or View — describes the procedure for linking to tables or views in external sources so that you can access the data in the same way you access any InterSystems IRIS object.
- The Link Procedure Wizard: Linking to a Stored Procedure — describes the procedure for linking to stored procedures in external sources.
- Controlling Gateway Connections — describes methods used to manage SQL Gateway connections.

1.1 Architecture of the InterSystems SQL Gateway

Internally, the InterSystems SQL Gateway uses the following components:

- The *Connection Manager* maintains a list of *logical connection definitions* for InterSystems IRIS. Each definition has a logical name used in InterSystems IRIS, as well as connection details for a specific external ODBC or JDBC compliant database. The InterSystems SQL Gateway uses these logical names when it establishes connections (see "Creating Gateway Connections for External Sources").
- The *InterSystems SQL Gateway API* is a set of functions used by an InterSystems IRIS program to communicate with a third-party RDBMS. These functions are implemented by means of a shared library, which is responsible for making the ODBC or JDBC calls.
- The *External Table Query Processor* is an extension to the InterSystems SQL Query Processor that handles queries targeted at external tables.
- The *SQL Dictionary* stores a list of all defined SQL tables. A given table is marked as "external" when its data is stored in a third-party RDBMS. When the InterSystems SQL Query Processor detects that the table (or tables) referenced within an SQL query are external, it invokes the External Table Query Processor, which generates a *query execution plan* by calling the InterSystems SQL Gateway API instead of accessing data stored within InterSystems IRIS.

1.1.1 Persisting External Tables in InterSystems IRIS

All object persistence in InterSystems IRIS is provided by means of a storage class (see "Storage Definitions and Storage Classes" in *Defining and Using Classes*), which generates the code needed to save and retrieve a persistent object within a database. The SQL storage class (%Storage.SQL) provides object persistence by means of specially generated SQL queries.

A class that uses %Storage.SQL for persistence indicates that it is an "external" class by providing a value for its CONNECTION and EXTERNALTABLENAME class parameters. The class compiler creates an SQL table definition for the class, and generates the SQL queries for the object persistence code. These queries automatically make calls to the correct external database by means of the External Table Query Processor.

1.1.2 Restrictions on SQL Gateway Queries

When you use the InterSystems SQL Gateway, note the following restrictions:

- All the tables listed in the FROM clause of an SQL query must come from the same data source. Queries that join data from heterogeneous data sources are not allowed.
- SQL queries targeted at external databases cannot use the following InterSystems SQL extensions:
  - The "->" operator.
  - The %EXACT function, or the %SYSTEM.Util **Collation()** method with the collation flag set to EXACT.
  - The inclusion of other columns within a count (*) query.
  - InterSystems IRIS-specific operators that have % as the first character of their name.

1.2 Creating Gateway Connections for External Sources

InterSystems IRIS maintains a list of SQL Gateway connection definitions, which are logical names for connections to external data sources. Each connection definition consists of a logical name (for use within InterSystems IRIS), information on connecting to the data source, and a user name and password to use when establishing the connection. These connections are stored in the table %Library.sys_SQLConnection. You can export data from this table and import it into another InterSystems IRIS instance.

Each gateway connection consists of the following details:

- A logical name for the gateway connection. This name would be used, for example, within any InterSystems SQL queries.
- Optional login credentials to access the database.
- Optional information to control the JDBC or ODBC driver.
- Driver-specific connection details:
  - For JDBC: The full class name of the JDBC client driver, the driver class path (a list of JAR files to search when locating the JDBC driver), and the JDBC connection URL.
  - For ODBC: a DSN (data source name), defined in the usual way (see Using an InterSystems Database as an ODBC Data Source on Windows and Using an InterSystems Database as an ODBC Data Source on UNIX® in *Using the InterSystems ODBC Driver*).

Note: When creating an SQL gateway connection for use by the Link Table Wizard using Microsoft SQL Server DNS configuration, do not set the **Use regional settings** option. This option is intended only for applications that display data, not for applications that process data.

See the following chapters for detailed information on creating logical connection definitions:

- Connecting with the JDBC Driver
- Connecting with the ODBC Driver

1.3 The Link Table Wizard: Linking to a Table or View

The Management Portal provides a wizard that you can use to link to an external table in an ODBC- or JDBC-compliant database. When you have linked to an external table, you can:

- Access data stored in third-party relational databases within InterSystems IRIS applications using objects and/or SQL queries.
- Store persistent InterSystems IRIS objects in external relational databases.

For example, suppose you have an Employee table stored within an external relational database. You can use this table within InterSystems IRIS as an object by creating an Employee class that communicates (by executing SQL queries via JDBC or ODBC) with the external database.

From the perspective of an InterSystems IRIS application, the Employee class behaves in much the same way as any other persistent class: You can open instances, modify, and save them. If you issue SQL queries against the Employee class, they are automatically dispatched to the external database.

The use of the InterSystems SQL Gateway is independent of application logic; an application can be modified to switch between external databases and the built-in InterSystems IRIS database with minimal effort and no change to application logic.

Any class that uses the InterSystems SQL Gateway to provide object persistence is identical in usage to classes that using native persistence and can make full use of InterSystems IRIS features including Java, SQL, and Web access.

1.3.1 Using the Link Table Wizard

When you link to an external table or view, you create a persistent InterSystems IRIS class that is linked to that table or view. The new class stores and retrieves data from the external source using the SQL Gateway. You can specify information about both the InterSystems IRIS class and the corresponding SQL table in InterSystems IRIS.

Note: This wizard generates ObjectScript code with class names and class member names that you control. When you use this wizard, be sure to follow the rules for ObjectScript identifiers, including length limits (see the section on Naming Conventions in *Defining and Using Classes*).

- If you have not yet created a gateway connection to the external database, do so before you begin (see "Creating Gateway Connections for External Sources").
- From the Management Portal select **System Explorer**, then **SQL**. Select a namespace with the **Switch** option at the top of the page; this displays the list of available namespaces.

  At the top of the page, click the `Wizards` drop-down list, and select **Link Table**.
- On the first page of the wizard, select one or more table or views, as follows:

  - Select a destination namespace — Select the InterSystems IRIS namespace to which the data will be copied.
  - Schema Filter — Specify a schema (class package) name that contains the table or view. You can specify a name with wildcards to return multiple schemas, or % to return all schemas. For example, C% will return all schemas in the namespace beginning with the letter C. Use of this filter is recommended, as it will shorten the return list of schemas to select from, and thus improve loading speed. You can select multiple items. In this case, when you click Next, the next screen prompts you for a package name. Specify the name of the package to contain the classes and then click Finish.
  - Table Filter — Specify the table or view to link to. You can specify a name with wildcards to return multiple tables and/or views, or % to return all tables/views.
  - Table type — Select TABLE, VIEW, SYSTEM TABLE, or ALL. The default is TABLE.
  - Select a SQL Gateway connection — Select the SQL Gateway connection to use.

- Click Next.
- On the second page, specify which fields should be available as object properties in InterSystems IRIS. Make changes as follows:
  - Highlight one or more fields and click the single arrow to move it or them from one list to another; click the double arrow to move all fields (selected or not) from one list to another.
  - In the selected list, use the up and down arrows to modify the order of the fields in the table that InterSystems IRIS projects for the given class. This does not affect the order of the properties in the class definition.
- Click Next.
- On the third page, specify information about the properties in the generated class. For each property, you can specify all the available options:
  - Read only — Select this check box to make the property read-only. This controls the ReadOnly keyword for the property.

    Tip: Use the select_all check box to select or clear all the check boxes in this column.
  - New Property Name — Specifies the name of the object property that will contain the data from this field.
  - New Column Name (SQL Field Name) — Specifies the SQL field name to use for this property. This controls the SqlFieldName keyword for the property.
- Click Next.
- On the last page, specify the following:
  - Primary Key — Select the primary key for the new InterSystems IRIS table from the list provided. In addition to the default key provided, you can click the "Browse" button to select one or more columns. You may select multiple columns; multiple columns are returned as a composite key separated by commas. You must specify a primary key.
  - New class name — Specify the name of the InterSystems IRIS class to create, including the package. The default package name is nullschema.
  - New table name — Specify the name of the SQL table to create in InterSystems IRIS. This controls the SqlTableName keyword for the class.
- Click Finish. The wizard displays the Background Jobs page with a link to the background tasks page.

- Click `Close`. Or click the given link to view the background tasks page. In either case, the wizard starts a background task to do the work.

The wizard stores a new class definition in the InterSystems IRIS database and compiles it. If data is present, it should be immediately visible in the external database (you can check by issuing SQL queries against the newly created InterSystems IRIS class/table). You can now use the new class as you would any other persistent class within InterSystems IRIS.

Note: **Closing the Link Table Connection**

By design, the code generated by the Link Table Wizard does not close the connections that it opens. This avoids problems such as conflicts between SQL statements that share the same connection. See "Controlling Gateway Connections" for more information.

1.3.2 Limitations When Using the Linked Table

As always, it is important to be aware of the particular limitations (syntactical or otherwise) and requirements of the database to which you are connecting. The following are a few examples:

- Informix: You cannot create a view inside of InterSystems SQL that is based on a linked Informix table, because the generated SQL is not valid in Informix.
- Sybase: As part of query processing, InterSystems SQL can transform the expression of an outer join into an equivalent canonicalized form. The SQL92-standard CROSS JOIN syntax may be required to reconstruct this form as SQL in order to access a linked table. Because Sybase does not support SQL92-standard CROSS JOIN, some queries using outer joins on linked Sybase tables will fail to execute.

Before you try to use a linked table, you might want to examine the cached query that is generated for it, to ensure that the syntax is valid for the database you are using. To see the cached query for a given linked table:

- In the Management Portal, go to **System Explorer**, **SQL**.
- Click the namespace you are interested in.
- Select the Schema from the pull-down list.
- Click `Cached Queries` for the package that contains the table. The system displays a table of the cached queries for this package. The `Query` column displays the full query.
- Optionally click the link for the query to see more details.

1.4 The Link Procedure Wizard: Linking to a Stored Procedure

The Management Portal provides a wizard that you can use to link to a stored procedure defined in an external ODBC- or JDBC-compliant database. When you link to the procedure, the system generates a method and a class to contain the method. When you link to an stored procedure, you create a class method that does the same action that the stored procedure does. This method is marked with the `SqlProc` keyword. The class method is generated within a new class, and you can specify information such as the class and package name. This method cannot accept a variable number of arguments. Default parameters are permitted, but the signature of the stored procedure is fixed.

Note: This wizard generates ObjectScript code with class names and class member names that you control. When you use this wizard, be sure to follow the rules for ObjectScript identifiers, including length limits (see the section on Naming Conventions in *Defining and Using Classes*).

- If you have not yet created a gateway connection to the external database, do so before you begin (see "Creating Gateway Connections for External Sources").
- From the Management Portal select **System Explorer**, then **SQL**. Select a namespace with the **Switch** option at the top of the page; this displays the list of available namespaces.

  At the top of the page, click the `Wizards` drop-down list, and select **Link Procedure**.
- On the first page of the wizard, select one or more procedures, as follows:
  - `Select a destination namespace` — Select the InterSystems IRIS namespace to which the data will be copied.
  - `Schema Filter` — Specify a schema (class package) name that contains the procedure. You can specify a name with wildcards to return multiple schemas, or % to return all schemas. For example, C% will return all schemas in the namespace beginning with the letter C. Use of this filter is recommended, as it will shorten the return list of schemas to select from, and thus improve loading speed.
  - `Procedure Filter` — Specify a procedure to link to. You can specify a name with wildcards to return multiple procedures, or % to return all procedures. You can select multiple procedures. In this case, when you click `Next`, the next screen prompts you for a package name. Specify the name of the package to contain the classes and then click `Finish`.
  - `Select a SQL Gateway connection` — Select the SQL Gateway connection to use.
- Click `Next`.
- On the second page, specify details about the class to generate in InterSystems IRIS:
  - `New package name` — Specify the name of the package to contain the class or classes.
  - `New class name` — Specify the name of the class to generate.
  - `New procedure name` — Specify the name of the procedure; specifically this controls the `SqlName` keyword of the method.
  - `New method name` — Specify the name of the method to generate.
  - `Description method name` — Optionally provide a description of the method; this is used as a comment for the class definition, to be displayed in the class reference.
- Click `Finish`. The wizard displays the Background Jobs page with a link to the background tasks page.
- Click `Close`. Or click the given link to view the background tasks page. In either case, the wizard starts a background task to do the work.

The wizard stores a new class definition within the InterSystems IRIS database and compiles it.

Note: **Closing the Link Procedure Connection**

By design, the code generated by the Link Procedure Wizard does not close the connections that it opens. This avoids problems such as conflicts between SQL statements that share the same connection. See "Controlling Gateway Connections" for more information.

1.5 Controlling Gateway Connections

In some cases, it may be necessary to manage connections created by code that links external tables or stored procedures (see "The Link Table Wizard" and "The Link Procedure Wizard"). SQL Gateway connections can be managed by the %SYSTEM.SQLGateway class, which provides methods such as the following:

- **DropAll()** — drop all open connections and unload the SQL Gateway library.
- **DropConnection()** — disconnect the specified JDBC or ODBC connection.
- **TestConnection()** — test a previously defined SQL Gateway connection (see "Creating Gateway Connections for External Sources") and write diagnostic output to the current device.
- Various methods for opening connections and controlling transactions. See the %SYSTEM.SQLGateway class documentation for full details.

These methods can be called with the special $SYSTEM object. For example, the following command would close a previously defined SQL Gateway connection named "MyConnectionName":

```
do $system.SQLGateway.DropConnection("MyConnectionName")
```

Note that SQL Gateway connection names are case-sensitive.

2

Connecting with the JDBC Driver

This chapter describes how to create a JDBC logical connection definition for the SQL Gateway. See *Using Java JDBC with InterSystems IRIS* for complete information on the InterSystems JDBC driver.

InterSystems IRIS maintains a list of SQL Gateway connection definitions, which are logical names for connections to external data sources. Each connection definition consists of a logical name (for use within InterSystems IRIS), information on connecting to the data source, and a username and password to use when establishing the connection. These connections are stored in the table %Library.sys_SQLConnection. You can export data from this table and import it into another InterSystems IRIS instance.

Note: **Controlling Gateway Logging and Java Version**

To monitor SQL Gateway problems, you can enable SQL Gateway logging (see "SQL Gateway Logging"). You can specify the version of Java to be used with the SQL Gateway by setting the InterSystems IRIS *JavaHome* parameter (see JavaHome in the *Parameter File Reference*).

2.1 Defining a Logical Connection in the Management Portal

To define a gateway connection for a JDBC-compliant data source, perform the following steps:

1. In the Management Portal, go to the **[System Administration] > [Connectivity] > [SQL Gateway Connections]** page.
2. Click **Create New Connection**.
3. On the **Gateway Connection** page, enter or choose values for the following fields:
   - For **Type**, choose **JDBC**.
   - **Connection Name** — Specify an identifier for the connection, for use within InterSystems IRIS.
   - **User** — Specify the name for the account to serve as the default for establishing connections, if needed.
   - **Password** — Specify the password associated with the default account.
   - **Driver name** — Full class name of the JDBC client driver.
   - **URL** — Connection URL for the data source, in the format required by the JDBC client driver that you are using.
   - **Class path** — Specifies a comma-separated list of additional JAR files to load.

- **Properties** — Optional string that specifies vendor-specific connection properties. If specified, this string should be of the following form:

  property= *value*; *property*= *value*;...

  See InterSystems JDBC Connection Properties for more information on connection properties.

For example, a typical connection might use the following values:

Setting	Value
Type	`JDBC`
Connection Name	`ConnectionJDBC1`
User	`JDBCUser`
Password	`JDBCPassword`
Driver name	`oracle.jdbc.driver.OracleDriver`
URL	`jdbc:oracle:thin:@//oraserver:1521/SID`
Class path	`/fill/path/to/ojdbc14.jar`
Properties	`oracle.jdbc.V8Compatibility=true; includeSynonyms=false;restrictGetTables=true`

For the other options, see "Implementation-specific Options."

4. Optionally test if the values are valid. To do so, click the **Test Connection** button. The screen will display a message indicating whether the values you have entered allow for a valid connection.
5. To create the named connection, click **Save**.
6. Click **Close**.

2.2 Creating a Connection between Namespaces

InterSystems IRIS provides JDBC drivers and can be used as a JDBC data source. That is, an InterSystems IRIS instance can connect to itself or to another InterSystems IRIS instance via JDBC and the SQL Gateway. Specifically, the connection is from a namespace in one InterSystems IRIS to a namespace in the other InterSystems IRIS. To connect in this way, you need the same information that you need for any other external database: the connection details for the database driver that you want to use. This section provides the basic information.

2.2.1 Connecting as a JDBC Data Source

To configure one Intersystems IRIS instance (IrisDB-1) to use another separate instance (IrisDB-2) as a JDBC data source, do the following:

1. Within IrisDB-1, use the SQL Gateway to create a JDBC connection to the namespace in IrisDB-2 that you want to use.
   - For **Type**, choose **JDBC**.
   - **Connection Name** — Specify an identifier for the connection, for use within IrisDB-1.
   - **User** — Specify the username needed to access IrisDB-2, if needed.

- **Password** — Specify the password for this user.
- **Driver name** — Use com.intersystems.jdbc.IRISDriver
- **URL** — Connection URL for the data source, in the following format:

  ```
  jdbc:IRIS://IP_address:port/namespace
  ```

  Here *IP_address:port* is the IP address and TCP port where IrisDB-2 is running, and *namespace* is the namespace to which you want to connect (see "Defining a JDBC Connection URL").

For example, a typical connection might use the following values:

Setting	Value
Type	JDBC
Connection Name	ConnectUser
User	_SYSTEM
Password	SYS
Driver name	com.intersystems.jdbc.IRISDriver
URL	jdbc:IRIS://127.0.0.1:51773/User

- **Class path** — Leave this blank.
- **Properties** — Optional string that specifies connection properties supported by the InterSystems JDBC drivers. If specified, this string should be of the following form:

  property= value; *property= value;...*
- **Conversion in composite Row IDs** — The default option should be suitable in most cases, but you can choose any option. The details are given in "Implementation-specific Options."
- **Do not use delimited identifiers by default** — The default (not to click this check box) should be suitable in most cases, but you can enable this option if desired. The details are given in "Implementation-specific Options."

2. Click **Save**.
3. Click **Close**.

2.3 Implementation-specific Options

Before you define an SQL gateway connection, you should make sure that you understand the requirements of the external database and of the database driver, because these requirements affect how you define the connection.

Do Not Use Delimited Identifiers by Default

The **Do not use delimited identifiers by default** option controls the format of identifiers in the generated routines.

Select this check box if you are using a database that does not support delimited SQL identifiers. This currently includes the following databases:

- Sybase
- Informix

- MS SQL Server

Clear the check box if you are using any other database. All SQL identifiers will be delimited.

Use COALESCE

The **Use COALESCE** option controls how a query is handled when it includes a parameter (?), and it has an effect only when a query parameter equals null.

- If you do not select **Use COALESCE** and if a query parameter equals null, the query returns only records that have null for the corresponding value. For example, consider a query of the following form:

  ```
  SELECT ID, Name from LinkedTables.Table WHERE Name %STARTSWITH ?
  ```

  If the provided parameter is null, the query would return only rows with null-valued names.

- If you select **Use COALESCE**, the query wraps each parameter within a `COALESCE` function call, which controls how null values are handled.

  Then, if a query parameter equals null, the query essentially treats the parameter as a wildcard. In the previous example, if the provided parameter is null, this query returns all rows.

Whether you select this option depends on your preferences and on whether the external database supports the `COALESCE` function.

To find out whether the external database supports the `COALESCE` function, consult the documentation for that database.

Conversion in Composite Row IDs

The **Conversion in composite Row IDs** option controls how non-character values are treated when forming a composite ID. Choose an option that is supported by your database:

- **Do not convert non-character values** — This option performs no conversion. This option is suitable only if your database supports concatenating non-character values to character values.
- **Use CAST** — This option uses`CAST` to convert non-character values to character values.
- **Use {fn convert ...}** — This option uses`{fn convert ...}` to convert non-character values to character values.

In all cases, the IDs are concatenated with`||` between the IDs (or transformed IDs).

Consult the documentation for the external database to find out which option or options it supports.

2.4 SQL Gateway Logging

The InterSystems SQL Gateway can generate a log when used with JDBC. To enable this logging:

- In the Management Portal, go to **[System Administration] > [Configuration] > [Connectivity] > [JDBC Gateway Settings]**.
- Specify a value for **JDBC Gateway Log**. This should be the name of a log file (for example, jdbcSqlGateway.log) that will record the interaction between the gateway and the database.

Note: Enable logging only when you need to perform troubleshooting. You should not enable logging during normal operation, because it will dramatically slow down performance.

3

Connecting with the ODBC Driver

This chapter describes how to create an ODBC logical connection definition for the SQL Gateway, and use the Data Migration Wizard. See *Using the InterSystems ODBC Driver* for complete information on how to use InterSystems ODBC.

The following topics are discussed:

- Creating ODBC Connections for External Sources — describes how to create an ODBC logical connection definition for the SQL Gateway.
- Using the Data Migration Wizard — describes how to migrate data from external ODBC sources and create an appropriate InterSystems IRIS class definition to store the data.
- Special Considerations for UNIX and Related Operating Systems — provides technical information about SQL Gateway drivers and the SQL Gateway test program.

3.1 Creating ODBC Connections for External Sources

InterSystems IRIS maintains a list of SQL Gateway connection definitions, which are logical names for connections to external data sources. Each connection definition consists of a logical name (for use within InterSystems IRIS), information on connecting to the data source, and a username and password to use when establishing the connection. These connections are stored in the table %Library.sys_SQLConnection. You can export data from this table and import it into another InterSystems IRIS instance.

The following topics are discussed in this section:

- Defining a Logical Connection in the Management Portal
- Creating an ODBC Connection through the SQL Gateway
- Implementation-specific Options

Note: For OS-specific instructions on how to create a DSN, see the following sections in *Using the InterSystems ODBC Driver*:

- "Using an InterSystems Database as an ODBC Data Source on Windows"
- "Using an InterSystems Database as an ODBC Data Source on UNIX®"

3.1.1 Defining a Logical Connection in the Management Portal

To define a gateway connection for an ODBC-compliant data source, perform the following steps:

1. Define an ODBC data source name (DSN) for the external database. See the documentation for the external database for information on how to do this.
2. In the Management Portal, go to the **[System Administration] > [Configuration] > [Connectivity] > [SQL Gateway Connections]** page (full menu path **[System Administration] > [Configuration] > [Connectivity] > [SQL Gateway Connections]**).
3. Click **Create New Connection**.
4. On the **Gateway Connection** page, enter or choose values for the following fields:
   - For **Type**, choose **ODBC**.
   - **Connection Name** — Specify an identifier for the connection, for use within InterSystems IRIS.
   - **Select an existing DSN** — Choose the DSN that you previously created. You must use a DSN, since the ODBC SQL Gateway does not support connections without a DSN.
   - **User** — Specify the name for the account to serve as the default for establishing connections, if needed.
   - **Password** — Specify the password associated with the default account.

   For example, a typical connection might use the following values:

Setting	Value
Type	`ODBC`
Connection Name	`ConnectionODBC1`
Select an existing DSN	`MyAccessPlayground`
User	`DBOwner`
Password	`DBPassword`

   For the other options, see "Implementation-specific Options" later in this section.
5. Optionally test if the values are valid. To do so, click the **Test Connection** button. The screen will display a message indicating whether the values you have entered in the previous step allow for a valid connection.
6. To create the named connection, click **Save**.
7. Click **Close**.

3.1.2 Creating an ODBC Connection through the SQL Gateway

InterSystems IRIS provides ODBC drivers and thus can be used as an ODBC data source. That is, an InterSystems IRIS instance can connect to itself or to another InterSystems IRIS instance via ODBC and the SQL Gateway. Specifically, the connection is from a namespace in one InterSystems IRIS to a namespace in the other InterSystems IRIS. To connect in this way, you need the same information that you need for any other external database: the connection details for the database driver that you want to use. This section provides the basic information.

To configure an InterSystems IRIS instance (*InterSystems IRIS_A*) to use another InterSystems IRIS instance (*InterSystems IRIS_B*) as an ODBC data source, do the following:

1. On the machine that is running *InterSystems IRIS_A*, create a DSN that represents the namespace in *InterSystems IRIS_B* that you want to use.

Tip: If *InterSystems IRIS_B* is installed on this machine, a suitable DSN might already be available, because when you install InterSystems IRIS, the installer automatically creates DSNs.

2. Within *InterSystems IRIS_A*, use the SQL Gateway to create an ODBC connection that uses that DSN. Provide the following details:
   - For **Type**, choose **ODBC**.
   - **Connection Name** — Specify an identifier for the connection, for use within *InterSystems IRIS_A*.
   - **Select an existing DSN** — Choose the DSN that you previously created for *InterSystems IRIS_B*.

   For example, a typical connection might use the following values:

Setting	Value
Type	`ODBC`
Connection Name	`TestConnection`
Select an existing DSN	`TestConnection`

   Tip: You do not need to specify **User** and **Password** because that information is part of the DSN itself.

3. Click **Save**.
4. Click **Close**.

3.1.3 Implementation-specific Options

Before you define an SQL gateway connection, you should make sure that you understand the requirements of the external database and of the database driver, because these requirements affect how you define the connection. The following options do not apply to all driver implementations.

Legacy Outer Join

The **Enable legacy outer join syntax (Sybase)** option controls whether the SQL gateway connection will enable you use to use legacy outer joins. Legacy outer joins use SQL syntax that predates the SQL-92 standard. To find out whether the external database supports such joins, consult the documentation for that database.

Needs Long Data Length

The **Needs long data length** option controls how the SQL gateway connection will bind data. The value of this option should agree with the `SQL_NEED_LONG_DATA_LEN` setting of the database driver. To find the value of this setting, use the ODBC **SQLGetInfo** function. If `SQL_NEED_LONG_DATA_LEN` equals `Y`, then select the **Needs long data length** option; otherwise clear it.

Supports Unicode Streams

The **Supports Unicode streams** option controls whether the SQL gateway connection supports Unicode data in streams, which are fields of type `LONGVARCHAR` or `LONGVARBINARY`.

- Clear this check box for Sybase. If you are using a Sybase database, all fields you access via the SQL gateway should include only UTF-8 data.
- Select this check box for other databases.

Do Not Use Delimited Identifiers by Default

The **Do not use delimited identifiers by default** option controls the format of identifiers in the generated routines.

Select this check box if you are using a database that does not support delimited SQL identifiers. This currently includes the following databases:

- Sybase
- Informix
- MS SQL Server

Clear the check box if you are using any other database. All SQL identifiers will be delimited.

Use COALESCE

The **Use COALESCE** option controls how a query is handled when it includes a parameter (?), and it has an effect only when a query parameter equals null.

- If you do not select **Use COALESCE** and if a query parameter equals null, the query returns only records that have null for the corresponding value. For example, consider a query of the following form:

  ```
  SELECT ID, Name from LinkedTables.Table WHERE Name %STARTSWITH ?
  ```

  If the provided parameter is null, the query would return only rows with null-valued names.

- If you select **Use COALESCE**, the query wraps each parameter within a `COALESCE` function call, which controls how null values are handled.

  Then, if a query parameter equals null, the query essentially treats the parameter as a wildcard. In the previous example, if the provided parameter is null, this query returns all rows, which is consistent with the behavior of typical ODBC clients.

Whether you select this option depends on your preferences and on whether the external database supports the `COALESCE` function.

To find out whether the external database supports the `COALESCE` function, consult the documentation for that database.

Conversion in Composite Row IDs

The **Conversion in composite Row IDs** option controls how non-character values are treated when forming a composite ID. Choose an option that is supported by your database:

- **Do not convert non-character values** — This option performs no conversion. This option is suitable only if your database supports concatenating non-character values to character values.
- **Use CAST** — This option uses `CAST` to convert non-character values to character values.
- **Use {fn convert ...}** — This option uses `{fn convert ...}` to convert non-character values to character values.

In all cases, the IDs are concatenated with `||` between the IDs (or transformed IDs).

Consult the documentation for the external database to find out which option or options it supports.

3.2 Using the Data Migration Wizard

The Management Portal provides a wizard that you can use to migrate data from an external table or view.

When you migrate data from a table or view in an external source, the system generates a persistent class to store data of that table or view and then copies the data. This wizard assumes that the class should have the same name as the table or view from which it comes; similarly, the property names are the same as in the table or view. After the class has been generated, it does not have any connection to external data source.

- If you have not yet created an SQL Gateway connection to the external database, do so before you begin (see "Creating Gateway Connections for External Sources").
- From the Management Portal select **System Explorer**, then **SQL**. Select a namespace with the **Switch** option at the top of the page; this displays the list of available namespaces.

  At the top of the page, click the `Wizards` drop-down list, and select **Data Migration**.
- On the first page of the wizard, select the table or view, as follows:
  - `Select a destination namespace` — Select the InterSystems IRIS namespace to which the data will be copied.
  - `Schema Filter` — Specify a schema (class package) name that contains the table or view. You can specify a name with wildcards to return multiple schemas, or % to return all schemas. For example, C% will return all schemas in the namespace beginning with the letter C. Use of this filter is recommended, as it will shorten the return list of schemas to select from, and thus improve loading speed.
  - `Table Filter` — Specify a table or view name. You can specify a name with wildcards to return multiple tables and/or views, or % to return all tables/views.
  - `Table type` — Select `TABLE`, `VIEW`, `SYSTEM TABLE`, or `ALL`. The default is `TABLE`.
  - `Select a SQL Gateway connection` — Select the SQL Gateway connection to use.
- Click `Next`.
- On the next page, you can optionally specify the following information for each class:
  - `New Schema` — Specify the package to contain the class or classes. Be sure to follow the rules for ObjectScript identifiers, including length limits (see the section on Naming Conventions in *Defining and Using Classes*).

    Tip: To change the package name for all classes, type a value at the top of this column and then click `Change all`.
  - `Copy Definition` — Select this check box to generate this class, based on the table definition in the external source. If you have already generated the class, you can clear this check box.
  - `Copy Data` — Select this check box to copy the data for this class from the external source. When you copy data, the wizard overwrites any existing data in the InterSystems IRIS class.
- Click `Next`. The wizard displays the following optional settings:
  - `Disable validation` — If checked, data will be imported with `%NOCHECK` specified in the *restriction* parameter of the **INSERT** command.
  - `Disable journaling for the importing process` — If checked, journaling will be disabled for the process performing the data migration (not system-wide). This can make the migration faster, at the cost of potentially leaving the migrated data in an indeterminate state if the migration is interrupted by a system failure. Journaling is re-enabled at the end of the run, successful or not.

  - `Defer indices` — If checked, indices are built after the data is inserted. The wizard calls the class' **%SortBegin()** method prior to inserting the data in the table. This causes the index entries to be written to a temporary location for sorting. They are written to the actual index location when the wizard calls the **%SortEnd()** method after all rows have been inserted. Do not use Defer Indices if there are Unique indices defined in the table and you want the migration to catch any unique constraint violations. A unique constraint violation will not be caught if Defer Indices is used.
  - `Disable triggers` — If checked, data will be imported with `%NOTRIGGER` specified in the *restriction* parameter of the **INSERT** command.
  - `Delete existing data from table before importing` — If checked, existing data will be deleted rather than merged with the new data.

- Click `Finish`. The wizard opens a new window and displays the Background Jobs page with a link to the background tasks page. Click `Close` to start the import immediately, or click the given link to view the background tasks page. In either case, the wizard starts the import as a background task.
- In the Data Migration Wizard window, click `Done` to go back to the home page of the Management Portal.

3.2.1 Microsoft Access and Foreign Key Constraints

When you use the Data Migration Wizard with Microsoft Access, the wizard tries to copy any foreign key constraints defined on the Access tables. To do this, it queries the `MSysRelationships` table in Access. By default, this table is hidden and does not provide read access. If the wizard can't access `MSysRelationships`, it migrates the data table definitions to InterSystems SQL without any foreign key constraints.

If you want the utility to migrate the foreign key constraints along with the table definitions, set Microsoft Access to provide read access for `MSysRelationships`, as follows:

- In Microsoft Access, make sure that system objects are displayed.
- Click `Tools > Options` and select the setting on the `View` tab.
- Click `Tools > Security > User and Group Permissions`. Then select the `Read` check box next to the table name.

3.3 Special Considerations for UNIX® and Related Platforms

This section provides technical information about SQL Gateway drivers and the SQL Gateway test program.

3.3.1 SQL Gateway Drivers for UNIX® Systems

The <install-dir>/bin/ directory contains the following versions of the shared object used by the SQL Gateway. This enables you to connect from InterSystems IRIS to other ODBC client drivers. These files are not installed if you perform a standalone installation.

linked against iODBC

- cgate.so — supports 8-bit ODBC.
- cgateiw.so — supports Unicode ODBC.

linked against unixODBC

- cgateu.so — supports 8-bit ODBC.
- cgateur64.so — supports 8-bit ODBC for 64-bit unixODBC

For more information, see "Using an InterSystems Database as an ODBC Data Source on UNIX®" in *Using the InterSystems ODBC Driver*.

Note: **Setting the Shared Library Path on UNIX® Systems**

When using third-party shared libraries on a UNIX® system, `LD_LIBRARY_PATH` must be defined by setting the InterSystems IRIS *LibPath* parameter (see "LibPath" in the *Configuration Parameter File Reference*). This is a security measure to prevent unprivileged users from changing the path.

3.3.2 Using the UNIX® ODBC SQL Gateway Test Program

Within a full UNIX® InterSystems IRIS installation, you can use a special program to test gateway access from InterSystems IRIS. In gatewaytest.sh, the InterSystems IRIS process making the initial call is the client application. Typically, a Gateway call from InterSystems IRIS calls the DSN of another vendor's database.

Note: The test program uses the default 8-bit iODBC–compliant drivers (libirisodbc.so and odbcgateway.so). For a complete list of the InterSystems ODBC client drivers and InterSystems SQL Gateway drivers available for supported UNIX® platforms, see "Using an InterSystems Database as an ODBC Data Source on UNIX®" in *Using the InterSystems ODBC Driver*.

The gateway test program consists of files in the directory *install-dir*/dev/odbc/samples/sqlgateway

- gatewaytest.sh — The shell script that runs the test. This script defines the *ODBCINI* environment variable (so that the ODBC initialization file can be found), sets up the search path to find the driver manager, and then accesses a DSN named `sampleodbc`, and executes a routine. This DSN is defined in the sample ODBC initialization file and points to the InterSystems IRIS USER namespace.
- SQLGatewayTest.ro — A routine that makes the callout to the InterSystems IRIS USER namespace using iODBC and the InterSystems ODBC client driver libirisodbc.so.

You may need to modify the shell script(gatewaytest.sh), depending on your configuration.

- The shell script is designed to work with Instance Authentication or unauthenticated modes and in Minimal or Normal security installations. It may need modification in other cases.
- By default, the shell script sets up the search paths to find the iODBC driver manager. You would change this if you use the unixODBC driver manager or if you install iODBC in a non-default way.
- The script also assumes that the ODBC initialization file is in the *install-dir*/mgr directory. You should adjust the script as needed to find the ODBC initialization file on your system.

To use the test program:

1. Go to *install-dir*/dev/odbc/samples/
2. Execute the test script by typing the following:

```
./sqlgateway/gatewaytest.sh
```

The gatewaytest.sh script does the following:

1. It starts an InterSystems IRIS session and runs the routine **SQLGatewayTest** in the USER namespace.

2. This application routine then loads the default InterSystems SQL Gateway driver, odbcgateway.so, which is linked against the iODBC driver manager.
3. The driver manager loads the client driver using information from the ODBC initialization file.
4. The client driver then establishes a TCP/IP connection to port 51773 and is connected to the InterSystems IRIS USER namespace using the DSN definition from the ODBC initialization file.
5. The routine executes the following query:

   ```
   SELECT * FROM SAMPLE.PERSON
   ```

6. The routine then fetches the first ten rows of the result set.

4

Using the ODBC SQL Gateway Programmatically

If you require options that are not provided by the standard SQL Gateway wizards, you can use the %Library.SQLGatewayConnection class to call ODBC functions from ObjectScript. You can either execute a dynamic query (obtaining a result set) or you can perform low-level ODBC programming. The following topics are discussed in this chapter:

- FetchSamples Example — lists a simple program that opens a connection, runs a query, and accesses the result set.
- Creating and Using an External Data Set — demonstrates using %SQL.Statement methods to run queries and access data sets.
- Calling ODBC Functions Directly — demonstrates how to call ODBC query functions directly, rather than through %SQL.Statement.
- Quick Reference for %SQLGatewayConnection — provides details about the supported methods and properties.

To use this section, you should have some experience with ODBC — this book does not provide details on the ODBC functions. If you encounter any problems, you can monitor the gateway by enabling logging for both InterSystems IRIS and ODBC (see the "Logging and Environment Variables" chapter in *Using the InterSystems ODBC Driver*).

Note: In the rest of this chapter, %Library.SQLGatewayConnection is referred to by its abbreviated name, %SQLGatewayConnection.

4.1 FetchSamples Example

The following example provides a simple demonstration of how to open a connection, prepare and execute a query, and access the resulting data set. See the entries in "Quick Reference for %SQLGatewayConnection" for information on **Connect()**, **Disconnect()**, ConnectionHandle, and sqlcode. See the Quick Reference section on "Supported ODBC Function Calls" for a list of supported ODBC functions and the %SQLGatewayConnection methods that call them.

ClassMethod FetchSamples

```
ClassMethod FetchSamples()
{
   #include %occInclude
   //Create new Gateway connection object
   set gc=##class(%SQLGatewayConnection).%New()
   if gc=$$$NULLOREF quit $$$ERROR($$$GeneralError,"Cannot create %SQLGatewayConnection.")
```

```
    //Make connection to target DSN
    set pDSN="Cache Samples"
    set usr="_system"
    set pwd="SYS"
    set sc=gc.Connect(pDSN,usr,pwd,0)
    if $$$ISERR(sc) quit sc
    if gc.ConnectionHandle="" quit $$$ERROR($$$GeneralError,"Connection failed")

    set sc=gc.AllocateStatement(.hstmt)
    if $$$ISERR(sc) quit sc

    //Prepare statement for execution
    set pQuery= "select * from Sample.Person"
    set sc=gc.Prepare(hstmt,pQuery)
    if $$$ISERR(sc) quit sc
    //Execute statement
    set sc=gc.Execute(hstmt)
    if $$$ISERR(sc) quit sc
    //Get list of columns returned by query
    set sc=gc.DescribeColumns(hstmt, .columnlist)
    if $$$ISERR(sc) quit sc

    //display column headers delimited by ":"
    set numcols=$listlength(columnlist)-1  //get number of columns
    for colnum=2:1:numcols+1 {
       Write $listget($listget(columnlist,colnum),1),":"
    }
    write !

    //Return first 200 rows
    set sc=gc.Fetch(hstmt)
    if $$$ISERR(sc) quit sc
    set rownum=1
    while((gc.sqlcode'=100) && (rownum<=200)) {
       for ii=1:1:numcols {
          set sc=gc.GetData(hstmt, ii, 1, .val)
          write " "_val
          if $$$ISERR(sc) break
       }
       set rownum=rownum+1
       write !
       set sc=gc.Fetch(hstmt)
       if $$$ISERR(sc) break
    }

    //Close cursor and then disconnect
    set sc=gc.CloseCursor(hstmt)
    if $$$ISERR(sc) quit sc

    set sc=gc.Disconnect()
    quit sc
}
```

4.2 Creating and Using an External Data Set

To create and use a data set that queries an external database, do the following:

1. Create an instance of %SQLGatewayConnection via the **%New()** method.
2. Call the **Connect()** method of that instance, passing arguments that specify the ODBC data source name, as well as the username and password that are needed to log into that source, if necessary.

   The **Connect()** method has the following signature:

   ```
   method Connect(dsn, usr, pwd, timeout) as %Status
   ```

   Here *dsn* is the DSN for the data source, *usr* is a user who can log into that data source, *pwd* is the corresponding password, and *timeout* specifies how long to wait for a connection.
3. Create an instance of %SQL.Statement via the **%New()** method, providing the string argument `"%DynamicQueryGW:SQLGW"`.

Note: This is slightly different from the argument that you use with a typical dynamic query (`"%DynamicQuery:SQL"`).

4. Invoke the **Prepare()** method of the result set. The first argument should be a string that consists of a SQL query, the second argument should be omitted, and the third argument should be the instance of %SQLGatewayConnection.
5. Call the **Execute()** method of the result set, optionally providing any arguments in the order expected by the query. This method returns a status, which should be checked.

To use the result set, you generally examine it one row at a time. You use methods of %SQL.Statement to retrieve information such as the value in a given column. Typically you iterate through all the rows using **Next()**, as demonstrated in the following example:

Example

```
ClassMethod SelectAndWrite() as %Status
{
    Set gateway=##class(%SQLGatewayConnection).%New()
    Set sc=gateway.Connect("AccessPlayground","","")
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    Set res=##class(%SQL.Statement).%New("%DynamicQueryGW:SQLGW")
    Set sc=res.Prepare("SELECT * FROM PEOPLE",,conn)
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    Set sc=res.Execute()
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    While res.Next()
    { Write !,res.GetData(1)," ",res.GetData(2)," ",res.GetData(3)
    }
    Set sc=gateway.Disconnect()
    Quit sc
}
```

For more information on %SQL.Statement, see the chapter "Dynamic SQL" in *Using InterSystems SQL*. Also see the class documentation for %SQL.Statement.

4.3 Calling ODBC Functions Directly

If %SQL.Statement does not provide enough control, you can use the %SQLGatewayConnection class to access ODBC directly. It provides a set of methods that correspond to ODBC functions (see "Supported ODBC Function Calls"), as well as other utility functions. You can connect to and use an ODBC-compliant database and then perform low-level ODBC programming. The overall procedure is as follows:

1. Create an instance of %SQLGatewayConnection via the **%New()** method.
2. Call the **Connect()** method of that instance, passing arguments that specify the ODBC data source name, as well as the username and password that are needed to log into that source, if necessary.
3. Call the **AllocateStatement()** method and receive (by reference) a statement handle.
4. Call other methods of the gateway instance, using that statement handle as an argument. Most of these methods call ODBC functions.

The following simple example demonstrates this procedure. It is similar to the example in the previous section, but it uses the %SQLGatewayConnection versions of **Prepare()** and **Execute()** to call ODBC query functions **SQLPrepare()** and **SQLExecute()** directly, rather than using the %SQL.Statement methods:

Executing a query using %SQLGatewayConnection methods

```
ClassMethod ExecuteQuery(mTable As %String)
{
   set mDSN="DSNtest"
   set mUsrName="SYSDBA"
   set mUsrPwd="masterkey"

   // Create an instance and connect
   set gateway=##class(%SQLGatewayConnection).%New()
   set status=gateway.Connect(mDSN,mUsrName,mUsrPwd)
   if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
   set hstmt=""

   // Allocate a statement
  set status=gateway.AllocateStatement(.hstmt)
   if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()

   // Use %SQLGatewayConnection to call ODBC query functions directly
   set status=gateway.Prepare(hstmt,"SELECT * FROM "_mTable)
   if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
   set status=gateway.Execute(hstmt)

   if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
   quit gateway.Disconnect()
}
```

Note: **Null Values and Empty Strings**

When you use the methods described in this chapter, remember that InterSystems IRIS and SQL have the following important differences:

- In SQL, `""` represents an empty string.
- In InterSystems IRIS, `""` equals null.
- In InterSystems IRIS, `$char(0)` equals an empty string.

4.4 Quick Reference for %SQLGatewayConnection

- Overview of the %SQLGatewayConnection API
- %SQLGatewayConnection Methods and Properties
- Supported ODBC Function Calls

4.4.1 Overview of the %SQLGatewayConnection API

The %SQLGatewayConnection class provides properties and methods that you can use to manage the connection to the external data source, check status information, and get information about the ODBC shared library. The methods and properties covered in this reference are listed below, organized by usage (see "Supported ODBC Function Calls" for methods not listed here):

Managing the Connection

The %SQLGatewayConnection class provides properties and methods that you can use to manage the connection to the external data source.

- DSN — (%String property) Data source name of the ODBC-compliant data source to which you want to connect.
- User — (%String property) Username to log into the data source.

- Password — (%String property) Associated password
- ConnectionHandle — (%Binary property) The current connection handle to the ODBC-compliant data source.
- **Connect()** — Establishes a connection to a DSN.
- **GetConnection()** — Establishes a connection using configuration settings to determine the DSN, username, and password.
- **SetConnectOption()** — Invokes the ODBC function **SQLSetConnectAttr**.
- **Disconnect()** — Closes the connection.

Status and Query Methods

Most of the methods of %SQLGatewayConnection return a status, which you should check. Status information is also available via the following properties and methods:

- sqlcode — (%Integer property) Contains the SQL code return by the last call (if any).
- GatewayStatus — (%Integer property) Indicates the status of the last call.
- **GetLastSQLCode()** — Returns an SQL code for the last call if this call does not return an SQL code.
- **GatewayStatusGet()** — Returns an error code for the last call.

The following methods get rows from the result set:

- **FetchRows()** — Returns (by reference) a specified number of rows for the given connection handle.
- **GetOneRow()** — Returns (by reference) the next row for the given connection handle.

The following methods get and set the values of bound query parameters:

- **GetParameter()** — Returns (by reference) the current value of the indicated parameter.
- **SetParameter()** — Sets the value of a previously bound parameter.

Using the Shared Library

The %SQLGatewayConnection class provides properties and methods that you can call to get information about the shared library used by the ODBC SQL Gateway.

- DLLHandle — (%Binary property) Handle for the shared library, as currently in use. This is set when you connect.
- DLLName — (%String property) Name of the shared library currently in use. This is set when you connect.
- **GetGTWVersion()** — Returns the current version of the shared library.
- **GetUV()** — Returns (by reference) whether the shared library was built as Unicode. Note that this method always returns a status of $$$OK.
- **UnloadDLL()** — Unloads the shared library from the process memory.

Note: The phrase *shared library* refers in general to the file or library that comprises the ODBC SQL Gateway. On Windows platforms, this is a file with the extension .dll. See "Special Considerations for UNIX® and Related Platforms" for more information on supported UNIX® shared objects. The properties and methods described here apply to all operating systems.

4.4.2 %SQLGatewayConnection Methods and Properties

This is an alphabetical listing of selected methods and properties. See "Supported ODBC Function Calls" for methods not listed here.

AllocateStatement()

Invokes ODBC function **SQLAllocHandle()** and creates the corresponding structures in the SQL Gateway.

```
method AllocateStatement(ByRef hstmt) as %Status
```

Connect()

Establishes a connection to a DSN.

```
method Connect(dsn, usr, pwd, timeout) as %Status
```

If username and password are both empty, this method calls the ODBC function **SQLDriverConnect()**. If that call is unsuccessful or username/password are specified, the method calls the ODBC function **SQLConnect()**.

If the timeout parameter is not 0, **SQLSetConnectAttr()** is first called to set `SQL_ATTR_LOGIN_TIMEOUT`.

ConnectionHandle property

%Binary property that provides the current connection handle to the ODBC-compliant data source.

Disconnect()

Closes the connection.

```
method Disconnect() as %Status
```

DLLHandle property

%Binary property that provides the handle for the shared library, as currently in use. This is set when you connect.

DLLName property

%String property that provides the name of the shared library currently in use. This is set when you connect.

DSN property

%String property that provides the data source name of the ODBC-compliant data source to which you want to connect.

FetchRows()

Returns (by reference) a specified number of rows for the given connection handle.

```
method FetchRows(hstmt, Output rlist As %List, nrows As %Integer) as %Status
```

Here *hstmt* is the connection handle, returned (by reference) from **AllocateStatement()**. Also, *rlist* is the returned list of rows; this is an InterSystems IRIS $list. Each item in the list contains a row. If there is no data (`SQL_CODE = 100`), fetching is assumed to be successful but the return list is empty.

CAUTION: This method is primarily useful for testing, and it truncates character fields up to 120 characters so that more fields would fit in a row. Use **GetData()** instead when you need non-truncated data.

GatewayStatus property

%String property that provides the status of the last call. Status value will be one of the following:

- 0 - success
- -1 - SQL error
- -1000 - critical error

GatewayStatusGet()

Returns an error code for the last call.

```
method GatewayStatusGet() as %Integer
```

It does not initialize the error code and can be called multiple times. See the previous notes for the GatewayStatus property.

GetConnection()

Establishes a connection, using configuration file entries to determine the DSN, user name, and password.

```
method GetConnection(conn, timeout) as %Status
```

GetGTWVersion()

Returns the current version of the shared library.

```
method GetGTWVersion() as %Integer
```

GetLastSQLCode()

Returns an SQL code for the last call if this call does not return an SQL code (for example, if you used **SQLGetData()**).

```
method GetLastSQLCode() as %Integer
```

GetOneRow()

Returns (by reference) the next row for the given connection handle.

```
method GetOneRow(hstmt, ByRef row) as %Status
```

Here *hstmt* is the connection handle, returned (by reference) from **AllocateStatement()**. Also, *row* is the returned row, an InterSystems IRIS $list. Each item in the list contains a field. If there is no data (SQL_CODE = 100), fetching is assumed to be successful but the return list is empty.

CAUTION: This method is primarily useful for testing, and it truncates character fields up to 120 characters so that more fields would fit in a row. Use **GetData()** instead when you need non-truncated data.

GetParameter()

Returns (by reference) the current value of the indicated parameter.

```
method GetParameter(hstmt, pnbr, ByRef value) as %Status
```

Here *hstmt* is the connection handle returned (by reference) from **AllocateStatement()** and *pnbr* is the ordinal number of the parameter.

GetUV()

Returns (by reference) whether the shared library was built as Unicode.

```
method GetUV(ByRef infoval) as %Status
```

Note that this method always returns a status of $$$OK.

Password property

%String property that provides the associated password.

SetConnectOption()

Invokes the ODBC function **SQLSetConnectAttr()**.

```
method SetConnectOption(opt, val) as %Status
```

Only integer values are supported. Integer values for the *opt* argument may be taken from the sql.h and sqlext.h header files.

SetParameter()

Sets the value of a previously bound parameter.

```
method SetParameter(hstmt, pvalue, pnbr) as %Status
```

Here *hstmt* is the connection handle returned (by reference) from **AllocateStatement()**, *pvalue* is the value to use, and *pnbr* is the ordinal number of the parameter. The parameters are stored in $list format. If the allocated buffer is not sufficient, a new buffer will be allocated.

sqlcode property

%Integer property that provides the SQL code returned by the last call (if any).

UnloadDLL()

Unloads the shared library for the ODBC SQL Gateway from the process memory.

```
method UnloadDLL() as %Status
```

User property

%String property that provides the username to log into the data source.

4.4.3 Supported ODBC Function Calls

The following table lists the supported ODBC functions and indicates which %SQLGatewayConnection methods are used to call those functions.

These methods do not have a detailed listing in this chapter. For details on the method arguments, actions, and return values, see the InterSystems Class Library reference for %SQLGatewayConnection.

Table 4–1: Calling ODBC Functions from %SQLGatewayConnection

ODBC Function	Method That Calls This Function
SQLAllocHandle	**AllocateStatement()**
SQLBindParameter	**BindParameter()**
SQLCloseCursor	**CloseCursor()**

ODBC Function	Method That Calls This Function
SQLColAttribute	**DescribeCols()**
SQLColumns	**Columns()**
SQLColumnsW	**ColumnsW()**
SQLDescribeCols	**DescribeCols()**
SQLDescribeParam	**DescribeParam()**
SQLDiagRec	**GetErrorList()**
SQLEndTran	**Transact()**
SQLExecute	**Execute()**
SQLFetch	**Fetch()**
SQLFreeHandle	**DropStatement()**
SQLFreeStmt	**UnbindParameters()**
SQLGetData	**GetData()**
SQLGetDataW	**GetDataW()**
SQLGetInfo	**GetInfo()**
SQLMoreResults	**MoreResults()**
SQLNumParams	**DescribeParameters()**
SQLParamData	**ParamData()**
SQLPrepare	**Prepare()**
SQLPrepareW	**PrepareW()**
SQLPrimaryKeys	**PrimaryKeys()**
SQLPrimaryKeys	**PrimaryKeysW()**
SQLProcedureColumns	**ProcedureColumns()**
SQLProcedureColumnsW	**ProcedureColumnsW()**
SQLProcedures	**Procedures()**
SQLPutData	**PutData()**
SQLPutDataW	**PutDataW()**
SQLRowCount	**RowCount()**
SQLSetConnectAttr	**SetConnectOption()** (see special considerations)
SQLSetStmtAttr	**SetStmtOption()**
SQLSpecialColumns	**SpecialColumns()**
SQLSpecialColumnsW	**SpecialColumnsW()**
SQLTables	**Tables()**
SQLTablesW	**TablesW()**