VisualAge for Java, Version 2.0

Data Access

Contents

Notices

Programming Interface Information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and Service Marks

Chapter 1. About Relational Database Access

VisualAge for Java supports access to relational databases through JDBC. You can access relational data in an applet or application by using the Data Access beans on the Visual Composition Editor beans palette. (If you have VisualAge for Java, Enterprise Edition, you can also access relational data through the Data Access Builder, see "When to use Data Access Builder" on page 3.)

The Data Access beans are a feature of Visual Age for Java, and comprise a Select bean and a DBNavigator bean. To use the Data Access beans, you must first add the Data Access beans feature to Visual Age for Java.

Select bean

The Select bean is a non-visual bean. Using the Select bean you can query a relational database. You can also insert, update or delete a row in the result set and commit the changes in the database.

When you use a Select bean, you specify properties pertinent to relational database access, for example, when a lock should be acquired for a row in a table. You also specify a query property that contains a connection alias and an SQL specification for the Select bean.

- A connection alias specifies the database connection characteristics for the Select bean, for example, the URL for the connection, and the user ID and password to be passed with the connection request.
- An SQL specification specifies an SQL statement for the Select bean. You can enter the SQL statement manually (an SQL editor is provided to do this) or you can use the SQL Assist SmartGuide, to help you visually compose the SQL statement.

When you create a connection alias or an SQL specification, you identify a database access class to hold the definition. The connection alias and SQL specification for a Select bean can be in the same database access class, but this is not a requirement. In addition, different Select beans can identify the same database connection alias or SQL specification. If different Select beans specify the same connection alias, they share the database connection associated with that connection alias. If one Select bean commits updates to a database, it commits all uncommitted updates made by any Select bean sharing the database connection.

Accessing relational data

The Select bean provides a set of methods for relational database access. For example it provides an execute method to execute the SQL statement, and an updateRow method to update a row in the database based on data in the current row of the result set. To access relational data using a Select bean, you connect an interface component to the Select bean. For example, you can make an event-to-method connection between the actionPerformed event for a button and the execute method of the Select bean. When the button is selected, the SQL statement associated with the Select bean is executed.

When you execute an SQL statement using a Select bean, it returns a result set. However, the actual number of rows fetched into memory (cache) is controlled by various properties of the Select bean. You can set the value of these properties and thus control how many rows are fetched. The properties determine:

- The maximum number of rows that can be fetched into the cache
- The size of a packet, that is, a set of rows
- The maximum number of packets allowed in the cache
- Whether all the rows in the result set are to be fetched into the cache, or only a subset of the result set. “All the rows” means the lesser of:
  - The maximum number of rows
  - The product of the packet size times the maximum number of packets allowed in the cache.

  A “subset of the result set” means only the number of rows needed to satisfy a request. For example, if the result set is 100 rows, but the application only displays 10 rows, only 10 rows are fetched into the cache.

You can display data in the result set by making a property-to-property connection between the appropriate source property of the Select bean and an appropriate target property of an interface component such as a text field. If you use the SQL Assist SmartGuide to compose the SQL statement, VisualAge for Java generates two bound properties for each data column in the SQL specification. One property is the data column in its specified data type, another is a String representation of the data column. So, for example, you can make a property-to-property connection between the String representation of a data column in the result set and the text property of a text field.

Many of the Select bean methods are designed to operate on the current row of the result set. When an SQL statement is executed using a Select bean, the first row of the result set is the current row. If a column value of the result set is connected to an interface component such as a text field, the column value in the current row is displayed in the text field.

The Select bean includes methods to change the current row, for example, one method makes the next row in the result set the current row. Furthermore, each column value property for the Select bean is a bound property, so that if the current row is changed, it changes the data displayed in any interface component connected to the bound property.

Inserting, updating and deleting data

The Select bean also provides methods that you can use to insert, update, and delete relational data. To perform these operations you must first use a Select bean to retrieve a result set. You then apply the changes to the current row of the result set.

There are various ways to use these methods. For example, one way to implement updates in an application is to make an event-to-script connection between an appropriate interface component, such as a button, and a script. The script accesses the value in another interface component such as a text field, and uses a Select bean method to set the value of the column in the current row of the result set based on the accessed value. The script also uses a Select bean method to update the database with the reset value in the current row.

DBNavigator bean

The DBNavigator bean is a visual bean that is used with a Select bean. The DBNavigator bean provides a set of buttons that execute the SQL statement for the associated Select bean; perform other relational database operations, such as commit updates to the database; and navigate rows in the result set.

The DBNavigator bean is customizable. You can specify which of the buttons in the set you want displayed (however you cannot control the order of buttons in the display). You do this by setting properties in the DBNavigator bean.

To use the DBNavigator bean you create a property-to-property connection between the *this* property of the Select bean and the *model* property of the DBNavigator bean. The *this* property refers to the whole object of the Select bean. The *model* property specifies that the DBNavigator bean will navigate the associated Select bean.

When to use Data Access Builder

The Enterprise version of VisualAge for Java includes an Enterprise Access tool called Data Access Builder. It lets you generate beans that perform operations beyond the scope of the data access beans. For example, using Data Access Builder, an applet or application can access rows returned by DB2 stored procedures. You can also use Data Access Builder to generate methods that will perform any action you want on the database contents.

Using Data Access beans at run time

After you develop an application that contains the Data Access beans you can deploy it for use. VisualAge for Java provides several files that are used with these applications at run time. These files, which are in *root*/eab/runtime20, where *root* is the VisualAge for Java root directory, contain the class files, in several formats, for the classes in the IBM Data Access Beans project. The classpath should be modified to contain one of these files when an application is deployed.

The files are:
- ivjdab.jar. A compressed JAR file for use in network applications
- ivjdab.zip. An uncompressed ZIP file for use in local applications

In addition, VisualAge for Java provides two run time directories. These directories contain the same classes that are in the JAR or ZIP file. The directories are automatically added to the classpath when VisualAge for Java is installed. This allows applications that use the Data Access beans to run on the machine in which VisualAge for Java is installed without further modification to the classpath.

The directories are:
- com.ibm.db
- com.ibm.ivj.db.uibeans

RELATED TASKS

"Chapter 3. Accessing Relational Data" on page 7

Chapter 2. Composing with Data Access Beans

VisualAge provides a set of beans that you can use for data access. The following topics explain how to use these beans:

- “Adding the Select Bean to the Visual Composition Editor Surface” on page 7
- “Editing Select Bean Properties” on page 8
- “Making a Query Specification” on page 8
- “Displaying and Navigating the Result Set” on page 33
- “Inserting, Updating, or Deleting Data” on page 35

Chapter 3. Accessing Relational Data

You can access relational data using Data Access beans. However, before you can use the beans you must:

- Add the Data Access Beans feature to VisualAge for Java. You can do this using the Quick Start window.
- Add the directory or JAR/ZIP file to the Workspace classpath, as appropriate for the JDBC driver class you select for database connection. You can do this using the Options window.

After you add the feature and set the classpath, you can access relational data by using Data Access beans in the Visual Composition Editor. This involves:

- “Adding the Select Bean to the Visual Composition Editor Surface”
- “Editing Select Bean Properties” on page 8
- “Executing a Select Bean” on page 32

You can also use the Data Access beans to perform the following operations:

- “Displaying and Navigating the Result Set” on page 33
- “Inserting, Updating, or Deleting Data” on page 35

RELATED TASKS

Using the Quick Start Window

Setting the Class Path

Adding the Select Bean to the Visual Composition Editor Surface

To use the Select bean, the Data Access Beans feature must be added to VisualAge for Java.

The Select bean is a nonvisual bean that you use to access data in a relational database. Start by adding the Select bean to the Visual Composition Editor surface as follows:

1. From the category drop-down menu in the Visual Composition Editor, select the **Database** category.
2. Select .
3. Move the mouse pointer to the location on the Visual Composition Editor surface where you want to place the Select bean.
4. Press and hold mouse button 1. Without releasing the mouse button, move the mouse pointer to position it precisely.
5. Release the mouse button. The Select bean is placed at the location of the mouse pointer.

RELATED TASKS

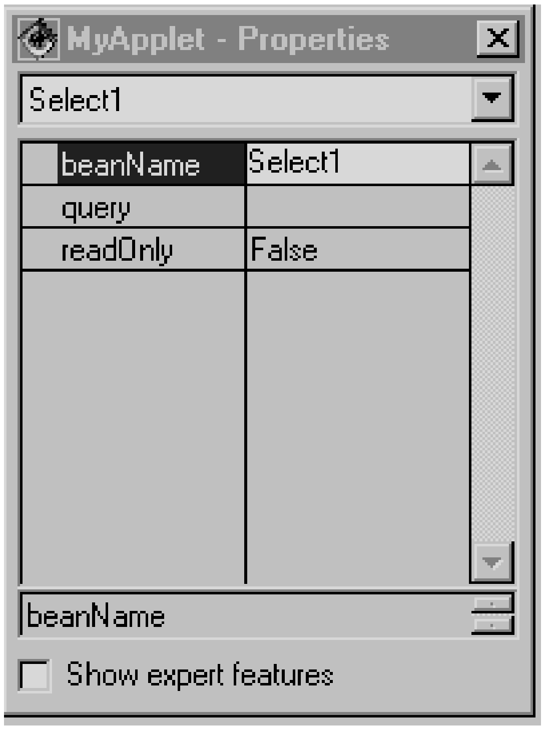
Adding a feature to VisualAge for Java

RELATED REFERENCES

“Chapter 6. Select” on page 41

Editing Select Bean Properties

You specify and control the operation of the Select bean by opening the property sheet for the Select bean and setting the value of the displayed properties.



beanName	Select1
query	
readOnly	False

beanName

You can control:

- The name of the Select bean. Specify the name as the **beanName** property value.
- The database connection characteristics and SQL statement. Specify these as a composite value for the **query** property.
- How many rows of the result are retrieved when the SQL statement is executed. The values of the **fillCacheOnExecute**, **maximumRows**, **packetSize**, and **maximumPacketsInCache** properties control this. These are properties displayed if the **Show Expert Features** checkbox is checked.
- When a lock is acquired for a row. Specify this in the **lockRows** property value. This property is displayed if the **Show Expert Features** checkbox is checked.
- Whether the result set is updatable. Specify this in the **readOnly** property value.

RELATED TASKS

"Making a Query Specification"

RELATED REFERENCES

"Chapter 6. Select" on page 41

Making a Query Specification

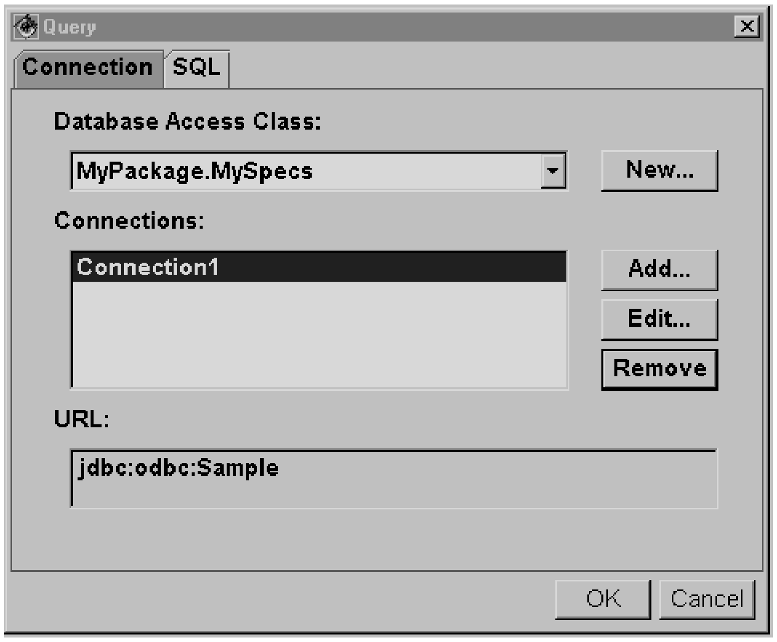
A query specification comprises a:

Connection alias
Specifies database connection characteristics for a Select bean

SQL specification
Specifies an SQL statement for a Select bean

To make a query specification:

1. Open the property sheet for the Select bean.
2. Select the **query** property value and click on the box in the value field. This opens the property editor for the query property.



Database Access Class: MyPackage.MySpecs

Connections: Connection1

URL: jdbc:odbc:Sample

3. Create a connection alias or select an existing connection alias.
4. Create an SQL specification or select an existing SQL specification.

RELATED TASKS

"Specifying a Connection Alias"

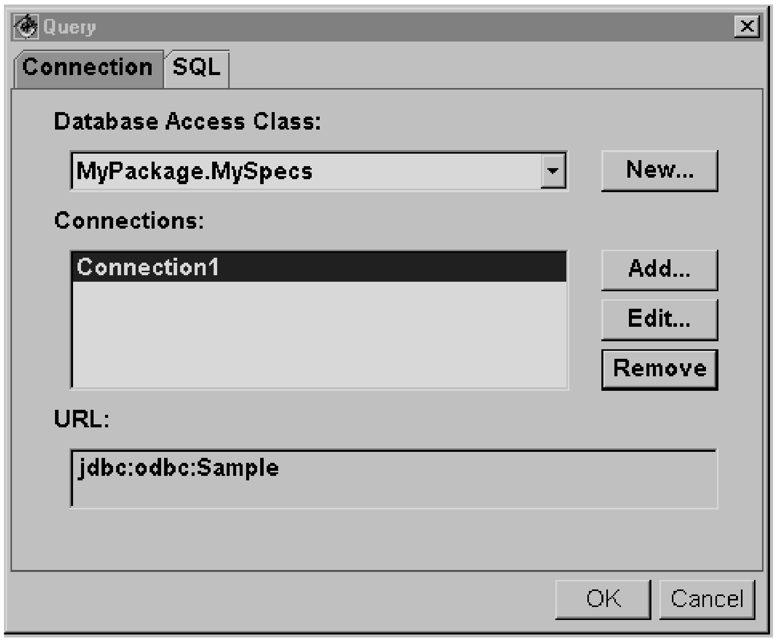
"Making an SQL Specification" on page 13

Specifying a Connection Alias

When you use a Select bean to access relational data, you must specify a connection alias for the Select bean. A connection alias specifies database connection characteristics for the Select bean. You can create a new connection alias or select an existing connection alias.

Use the Connection page of the Query property editor to specify a connection alias.

When you create a connection alias, you identify a database access class to hold the connection alias definition. Different Select beans can identify the same connection alias, and if so, they share the database connection associated with that connection alias. If one Select bean commits updates to a database, it commits all uncommitted updates made by any Select bean sharing the database connection.



1. Select a database access class from the drop-down list in the **Database Access Class** field. The list shows the database access classes that exist in the workspace. The format of each item in the list is *packagename*.*classname*, where *packagename* is the name of the package that contains the database access class, and *classname*, is the name of the class.

   To define a new database access class and add it to the **Database Access Class** list, select **New**.
2. Select a connection alias from the **Connections** list. The list identifies the connection aliases that are in the selected database access class. Selecting a connection alias from the list enables the SQL tab (select the SQL tab to make an SQL specification for the Select bean).

   To add a connection alias to the database access class, select **Add**.

   To edit the definition of a connection alias, select the connection alias name in the **Connections** list and then select **Edit**.

   To remove a connection alias from the **Connections** list, select it in the list and then select **Remove**.

When you finish specifying the connection alias, select **OK**.

To cancel specifying a connection alias, select **Cancel**.

RELATED TASKS

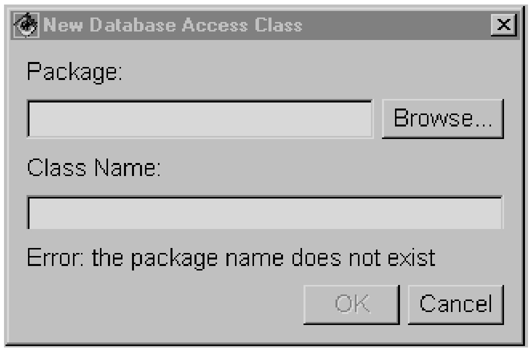
"Defining a Database Access Class" on page 11

"Defining or Editing a Connection Alias" on page 11

"Making an SQL Specification" on page 13

Defining a Database Access Class

Select **New** in the Connection page of the Query property editor to define a new database access class to contain a connection alias for a Select bean. Select **New** in the SQL page of the Query property editor to define a new database access class to contain an SQL specification for a Select bean. Selecting **New** opens the New Database Access Class window.



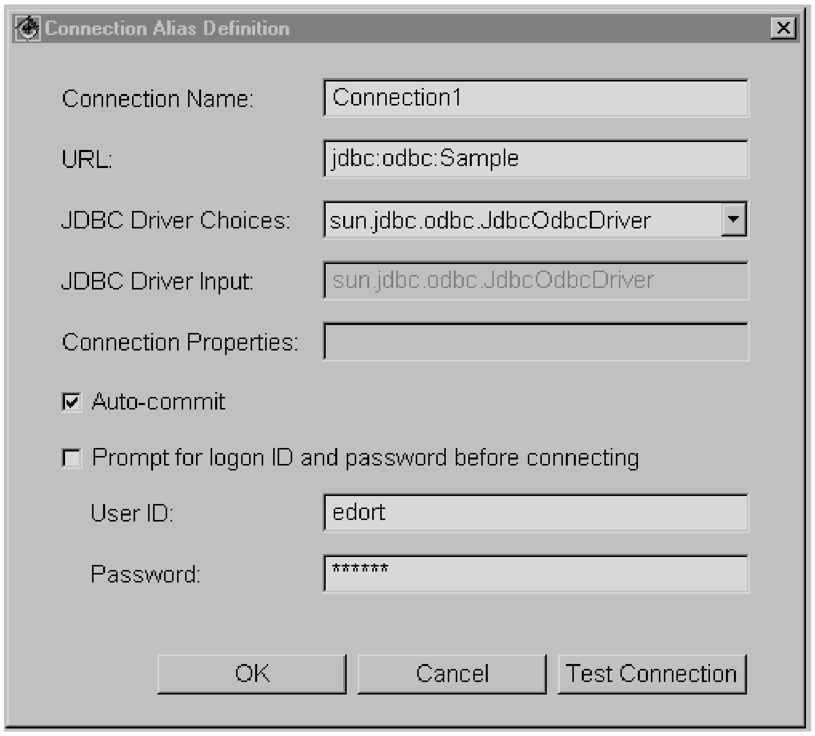
Error: the package name does not exist

1. Specify a package for the database access class in the **Package** field. Select **Browse** to view a list of the packages available in the workspace.
2. Specify a class name for the database access class in the **Class name** field.
3. When you finish specifying a package and class name, select **OK**. This creates a new database access class and adds it to the database access class list in the Connection page and SQL page of the Query property editor.

   To cancel defining a new database access class, select **Cancel**.

Defining or Editing a Connection Alias

A connection alias specifies the characteristics of a database connection for a Select bean. Select **Add** in the Connection page of the Query property editor to define a new connection alias. Select a connection alias from the **Connections** list in the Connection page of the Query property editor and select **Edit** to edit the selected connection alias. Selecting **Add** or **Edit** opens the Connection Alias Definition window.



Connection Name: Connection1

URL: jdbc:odbc:Sample

JDBC Driver Choices: sun.jdbc.odbc.JdbcOdbcDriver

JDBC Driver Input: sun.jdbc.odbc.JdbcOdbcDriver

Connection Properties:

☑ Auto-commit

☐ Prompt for logon ID and password before connecting

User ID: edort

Password: ******

- Specify a name for the connection alias in the **Connection Name** field. The name must be a valid Java method name.
- Specify in the **URL** field, the URL for the database connection. The URL specification must be in the format jdbc:*subprotocol*:*subname*, where *subprotocol* and *subname* identify the data source for the connection. The value of *subprotocol* depends on the JDBC driver used. For example, for the DB2 application JDBC driver, *subprotocol* is db2; for the Oracle thin driver, *subprotocol* is thin.

  The value of *subname* depends on the *subprotocol* specification; the *subname* value provides information to locate the database. For example, a full URL specification for an application accessing a local database named sample through the DB2 application JDBC driver is:

  ```
  jdbc:db2:sample
  ```

  Here, sample is the *subprotocol* value.

  By comparison, a full URL specification for an applet using the Sybase jConnect driver to access a database named sample, on a remote server named myserv, through port number 88 on the internet is:

  ```
  jdbc:sybase:Tds:myserver:88/sample
  ```

  Here, the *subprotocol* value includes the database server name, port number, and database name.
- Select a JDBC driver class from the **JDBC Driver Choices** list. For example, the DB2 application JDBC driver class is COM·ibm·db2·jdbc·app·DB2Driver. Select *Other driver* to specify a JDBC driver class that is not in list; then specify the JDBC driver class the **JDBC Driver Input** field.

The Workspace needs to have access to the JDBC driver class that you select. To ensure access, you need to add the directory or Jar/Zip file, as appropriate for the JDBC driver class, to the workspace classpath.

- Specify in the **Connection Properties** field, any properties to be passed in the database connection request, other than the user ID and password. Specify the properties in the following format: *prop*=*value*;*prop*=*value*;... where, *prop* is the name of the property, and *value* is the value of the property.

  In the following example, three properties are passed:

  ```
  proxy=myserver:88;a=1;b=2
  ```
- Check the **Auto-commit** checkbox if you want database updates to be automatically committed for each SQL query. If you do not check the checkbox, database updates are not automatically committed. This checkbox is checked by default.
- Check the **Prompt for logon ID password before connecting** checkbox if you want the user to be prompted for the user ID and password to be used in the database connection request. Do not check the checkbox if you want the user ID and password specified in the **User ID** and **Password** fields of the connection alias definition to be used in the database connection request.
- Specify in the **User ID** field, the user ID for the database connection request. This user ID is used if the **Prompt for logon ID password before connecting** checkbox is not checked.
- Specify in the **Password** field, the password for the database connection request. This password is used if the **Prompt for logon ID password before connecting** checkbox is not checked.

Select **Test Connection** to test the database connection using the specifications made in the connection alias definition.

When you finish defining or editing a connection alias, select **OK**. Selecting **OK** to define a new connection alias, creates a new method for the connection alias in the database access class and adds the connection alias to the **Connections** list in the Connection page of the Query property editor.

To cancel defining or editing a connection alias, select **Cancel**.

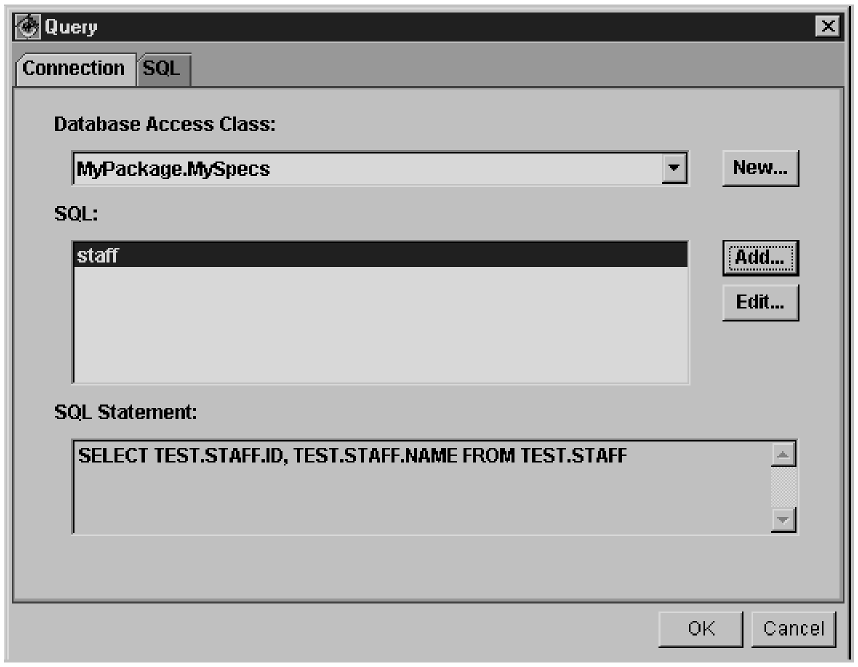
RELATED TASKS

Adding resources and paths to the class path

Making an SQL Specification

You use a Select bean to access relational data. Use the SQL page of the Query property editor to specify the SQL statement for the Select bean. You can create a new SQL specification or select an existing SQL specification. When you create an SQL specification, you have the option of composing it manually in an SQL editor or you can use the SQL Assist SmartGuide, to help you compose it.

When you create an SQL specification, you identify a database access class to hold the SQL specification definition. Different Select beans can identify the same SQL specification.



Database Access Class: MyPackage.MySpecs

SQL: staff

SQL Statement:

```
SELECT TEST.STAFF.ID, TEST.STAFF.NAME FROM TEST.STAFF
```

1. Select a database access class from the drop-down list in the **Database Access Class** field. The list shows the database access classes that exist in the workspace. The format of each item in the list is *packagename*.*classname*, where *packagename* is the name of the package that contains the database access class, and *classname*, is the name of the class.

   To define a new database access class and add it to the **Database Access Class** list, select **New**.
2. Select an SQL specification from the **SQL** list. The list identifies the SQL specifications that are in the selected database access class.

   To add an SQL specification to the database access class, select **Add**.

   To edit an SQL specification, select the SQL specifications name in the **SQL** list and then select **Edit**.

   To remove an SQL specification from the **SQL** list, select it in the list and then select **Remove**.

When you finish making the SQL specification, select **OK**.

To cancel making an SQL specification, select **Cancel**.

RELATED TASKS

"Defining a Database Access Class" on page 11

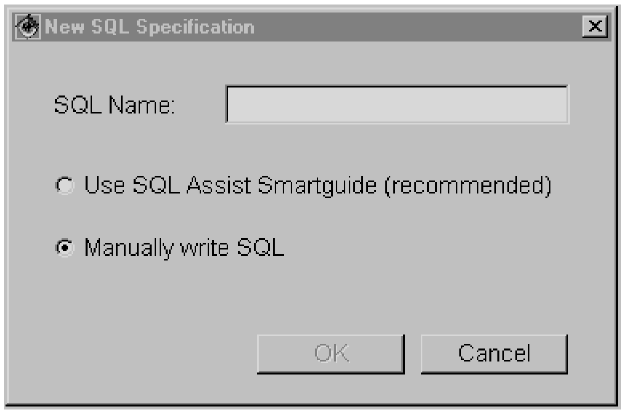
"Specifying a Connection Alias" on page 9

"Making a New SQL Specification"

"Composing or Editing an SQL Statement" on page 15

Making a New SQL Specification

Select **Add** in the SQL page of the Query property editor to make a new SQL specification for a Select bean. This opens the New SQL Specification window.



- Specify a name for the SQL specification in the **SQL Name** field. The name must be a valid Java method name.
- Select either the **Use SQLAssist SmartGuide** radio button or the **Manually write SQL** radio button.

  Selecting the **Use SQL Assist SmartGuide** radio button is recommended. SQL Assist is a SmartGuide that helps you visually compose an SQL statement.

  Selecting the **Manually write SQL** opens an SQL editor for you to enter an SQL statement.

When you finish, select **OK**. This opens the SQL Assist SmartGuide or the SQL editor, depending on the radio button selected.

To cancel making a new SQL specification, select **Cancel**.

RELATED TASKS

"Composing or Editing an SQL Statement"

Composing or Editing an SQL Statement

An SQL statement comprises the SQL specification for a Select bean. You can compose a new SQL statement or edit one that you already composed.

You compose a new SQL statement by selecting **Add** in the SQL page of the Query property editor. This opens the New SQL Specification window. You have two ways to compose the new SQL statement:
- Use the SQL Assist SmartGuide. The SQL Assist SmartGuide helps you build an SQL statement visually.
- Use an SQL editor to enter the SQL statement manually.

You are prompted to specify one of these SQL composition approaches in the New SQL Specification window.

You make a request to edit an SQL statement by selecting **Edit** in the SQL page of the Query property editor. This opens either the SQL Assist SmartGuide or the SQL editor, depending on which one you used to initially compose the SQL statement.

RELATED TASKS

"Composing an SQL Statement Visually" on page 16

“Composing an SQL Statement Manually”

Composing an SQL Statement Visually

Use the SQLAssist SmartGuide to visually compose an SQL statement for a Select bean. Using the SQLAssist SmartGuide you can:

- Specify the tables to be accessed in the SQL statement. (Required.)
- Join the tables. (Optional.)
- Specify search conditions for the SQL statement. (Optional.)
- Specify the table columns to return in the result set. (Optional.)
- Sort the result set. (Optional.)
- Remap a result column to a different Java class. (Optional.)
- Display the resulting SQL statement. (Optional.)

When you display the SQL statement, you can also run it as a test using the SQL Assist SmartGuide. You can also copy the SQL statement to the clipboard or save it.

RELATED TASKS

“Specifying the Tables for an SQL Statement” on page 17

“Joining Tables” on page 20

“Specifying Search Conditions” on page 23

“Specifying Result Columns” on page 26

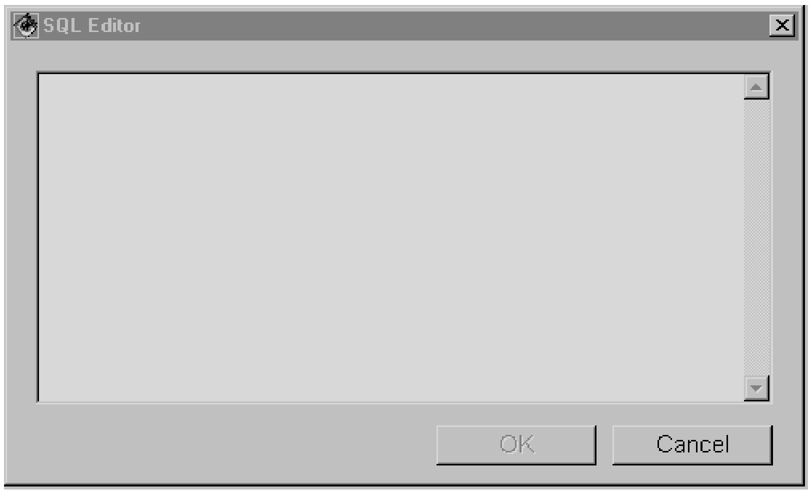
“Sorting the Result Set” on page 28

“Remapping Data to a Different SQL Data Type” on page 29

“Displaying the SQL statement” on page 31

Composing an SQL Statement Manually

Use the SQL Editor to manually define an SQL statement for a Select bean. You also use the SQL Editor to edit an SQL statement that you previously created using the SQL Editor. (An SQL statement created with the SQL Assist SmartGuide is edited using the SQL Assist SmartGuide.)



As you compose your SQL statement, you can:

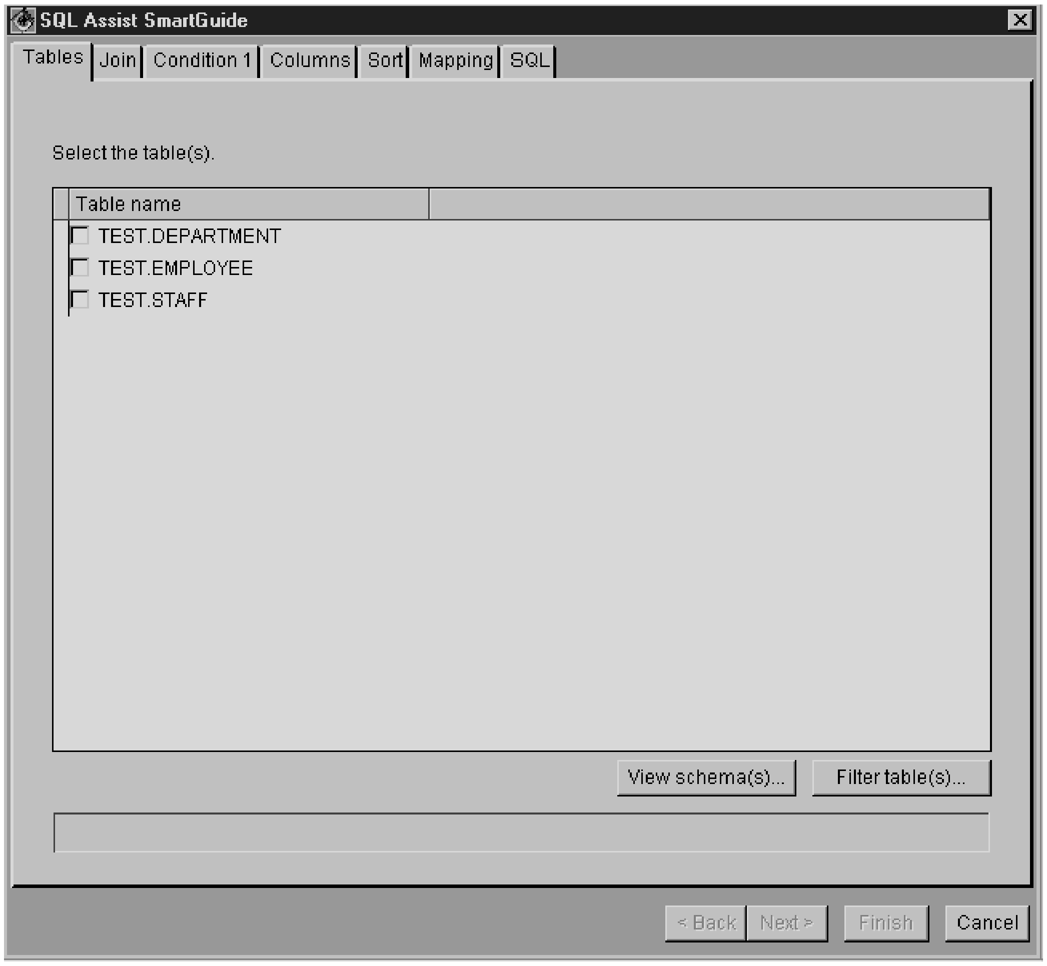
- Cut text from the SQL statement. Select the text in the SQL Editor and select the **Ctlr+X** keys on the keyboard. This copies the selected text to the clipboard. The cut text is available for pasting.
- Copy text from the SQL statement to the clipboard. Select the text in the SQL Editor and select the **Ctrl+C** keys on the keyboard. The copied text is available for pasting.
- Paste text from the clipboard into the SQL statement. Select the **Ctlr+V** keys on the keyboard. This pastes the text at the position of the cursor.
- Delete text from the SQL statement. Select the text in the SQL Editor and select the **Delete** or the **Backspace** key on the keyboard.

When you finish composing the SQL statement, select **OK**.

To cancel composing the SQL statement, select **Cancel**.

Specifying the Tables for an SQL Statement

Use the Tables page of the SQLAssist SmartGuide to specify the tables that are accessed in an SQL statement. The table names that you select in the Tables page will appear in the FROM clause of the SQL statement.



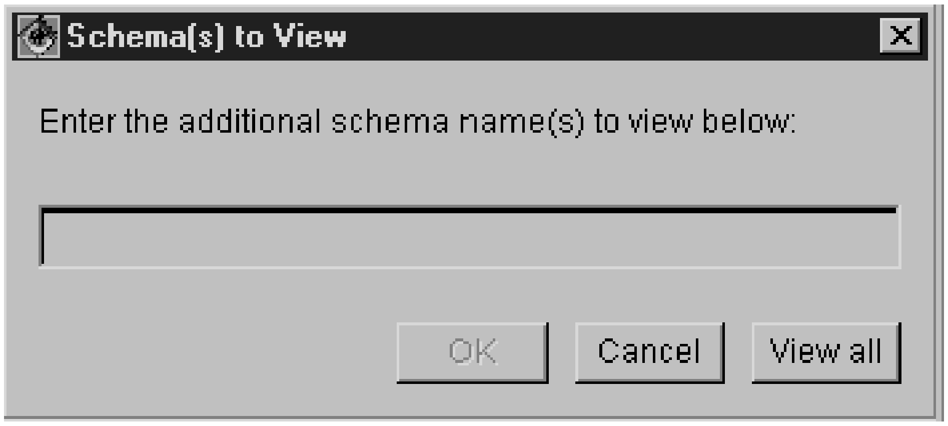
Select the table(s).

Table name
☐ TEST.DEPARTMENT
☐ TEST.EMPLOYEE
☐ TEST.STAFF

The **Table name** field list the tables that you can access in the database identified by the currently selected connection alias. By default, the tables listed are those whose schema is the user ID specified for the database connection. If no tables in the database have that schema, all the tables in the database are listed.

You can add table names to those displayed in the list, by selecting **View schema(s)**. You can filter the table names displayed in the list by selecting **Filter table(s)**.

Selecting **View schema(s)** opens the Schema(s) to View window.

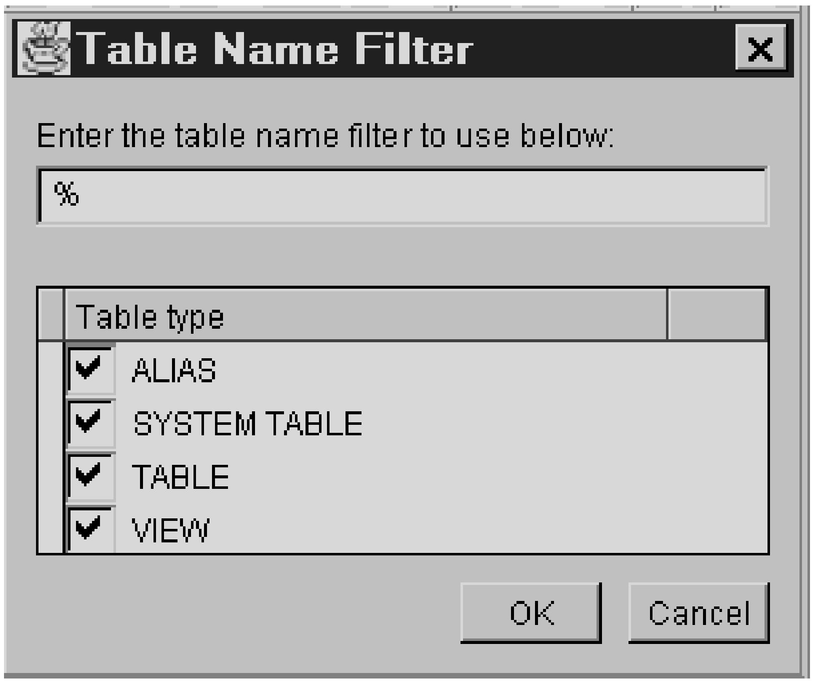


You are then prompted to enter additional schemas for the table name list. For example, if you enter a schema JOE, all tables that have the schema JOE in the database will be added to the table names already listed.

The % character is a wildcard character. For example, specifying %OE requests the display of all table names that are in a schema that ends with the characters OE. The specification J%OE requests the display of all tables names that are in a schema that begins with J and ends with OE.

You can display all the tables in the database by selecting **View all**.

Selecting **Filter table(s)** opens the Table Name Filter window.



You are then prompted to enter the following information for the filter:

- Filtering characters for the table name. The filtering characters are case-sensitive. These characters limit the display to only table names beginning with those characters. For example, if you enter EMP, only table names that begin with EMP are listed, such as the EMPLOYEE table. The % character is a wildcard character. Use it to position the filter. For example, specifying %ID requests the display of all table names that end with the characters ID. The specification N%ID requests the display of all tables names that begin with N and end with ID.

- Table types. This determines the type of tables that will be displayed in the Tables page. You can specify alias tables, system tables, user table, or views by checking its checkbox in the **Table type** field. You can check multiple table types.

When you finish entering information that adds or filters table names in the list, select **OK**. This closes the Schema(s) to View or Table Name Filter window, as appropriate.

Select a table for the SQL statement by checking the checkbox next to the table name in the **Table name** list. You can select more than one table.

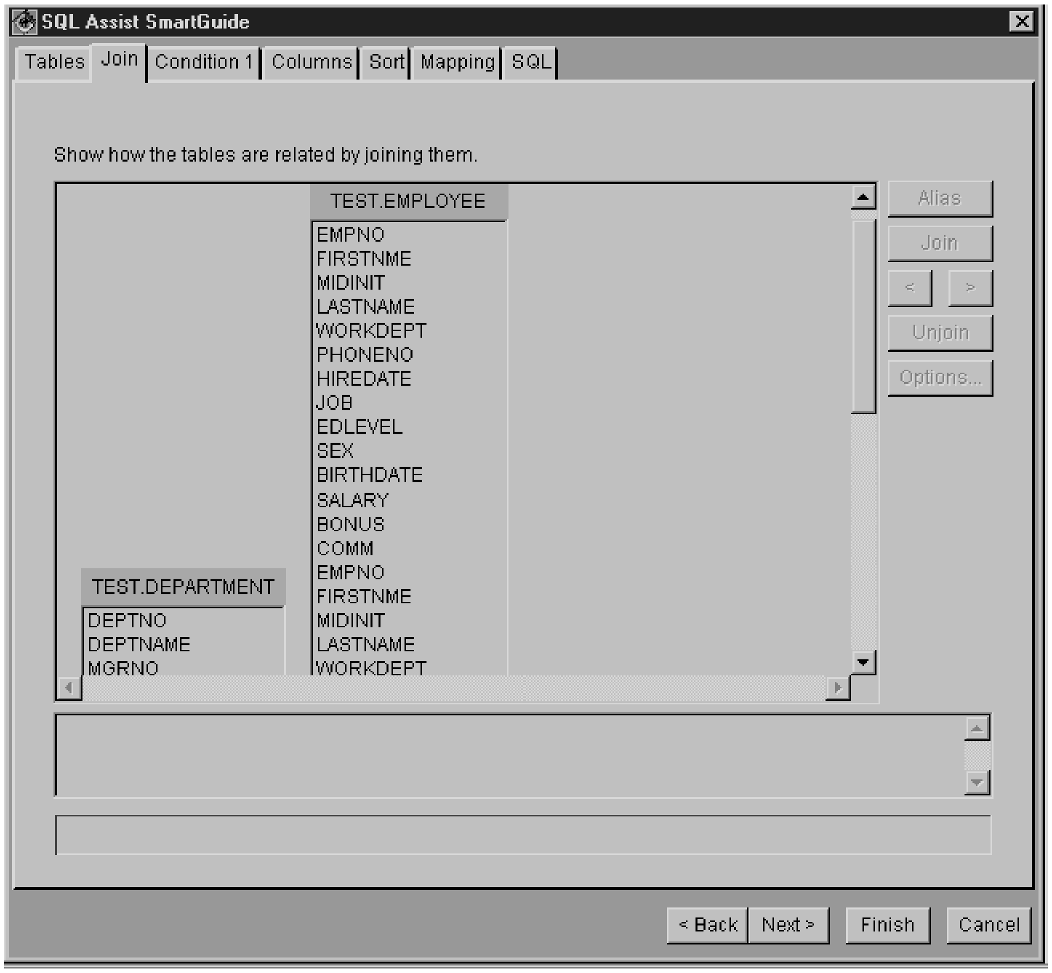
When you finish, select **Next**. This displays the Join page of the SQL Assist SmartGuide. Use this page to specify table joins for the SQL statement. You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQL Assist SmartGuide.

To cancel visual composition of the SQL statement, select **Cancel**.

Joining Tables

Use the Join page of the SQLAssist SmartGuide to join tables in an SQL statement. The join page displays the columns of each table selected in the Tables page.



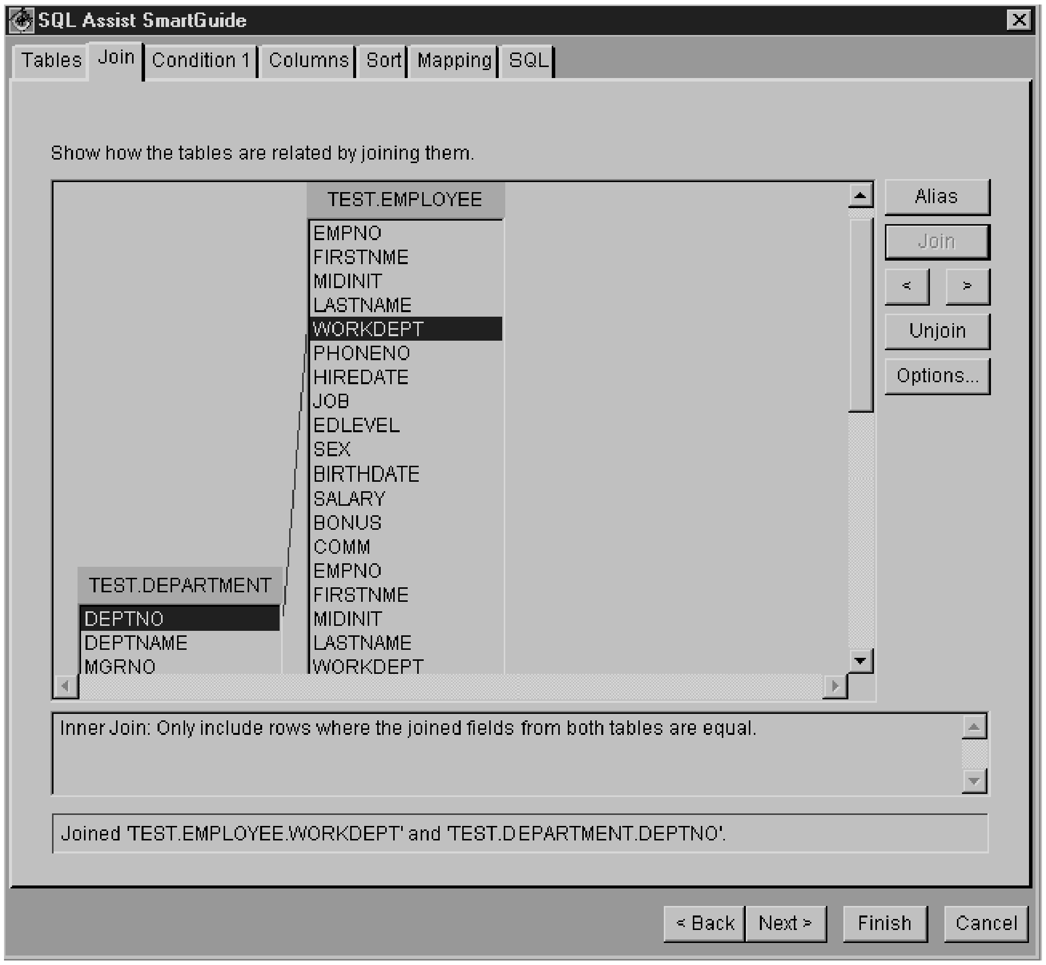
Show how the tables are related by joining them.

TEST.EMPLOYEE
EMPNO
FIRSTNME
MIDINIT
LASTNAME
WORKDEPT
PHONENO
HIREDATE
JOB
EDLEVEL
SEX
BIRTHDATE
SALARY
BONUS
COMM
EMPNO
FIRSTNME
MIDINIT
LASTNAME
WORKDEPT

TEST.DEPARTMENT
DEPTNO
DEPTNAME
MGRNO

Requesting a join

1. Select one of the displayed columns in a table. The informational area indicates the status of the join.
2. Select a displayed column in another table. A line appears connecting the columns to indicate the requested join. Notice that the information area is updated. The information area also indicates if a requested join is invalid (for instance, because of a mismatch in the data type of the columns). The control buttons on the join page are also enabled.



Show how the tables are related by joining them.

TEST.EMPLOYEE: EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, COMM, EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT

TEST.DEPARTMENT: DEPTNO, DEPTNAME, MGRNO

Inner Join: Only include rows where the joined fields from both tables are equal.

Joined 'TEST.EMPLOYEE.WORKDEPT' and 'TEST.DEPARTMENT.DEPTNO'.

By default, a join request is assumed to be an inner join. An inner join requests only the rows where the values of the two columns match. You can also request other types of joins by selecting **Options**. You can select:

- Left outer join. This is a request for an inner join and any additional rows in the left table (as viewed in the Join page) that are not already included in the inner join.
- Right outer join. This is a request for an inner join and any additional rows in the right table (as viewed in the Join page) that are not already included in the inner join.

Inner join is also an option. You can choose this if want to change a join type from a left or right outer join.

3. Select **Join**. The color of the join line changes to red indicating that the join is enabled.

Requesting additional joins

You can request additional joins in the same way as the initial join. You can join other displayed columns in the same tables or in other tables. If you request multiple joins, you can navigate between the joins by selecting **>** or **<**. The selected join is indicated by a red join line.

Joining a table alias

You can join a column in a table with a column of an alias. An alias is an alternate name for a table. Joining a column to an alias gives you a way of joining two columns in the same table. You can do this as follows:

1. Select the column for the join in the table.
2. Select **Alias**. This creates an alias for the selected table and displays its columns. These are the same columns as the associated table.
3. Select a column in the alias.
4. Select **Options** if you want to change the type of join.
5. Select **Join**

Removing a join

To remove a join, select the joined columns or navigate to the pertinent join. Then select **Unjoin**. The join line is removed.

When you finish the join

When you finish the join specification, select **Next**. This displays the Condition 1 page of the SQLAssist SmartGuide. Use this page to specify a search condition for the SQL statement.

Select **Back** to display the Tables page of the SQLAssist SmartGuide. If appropriate, you can then change the tables selected for the query.

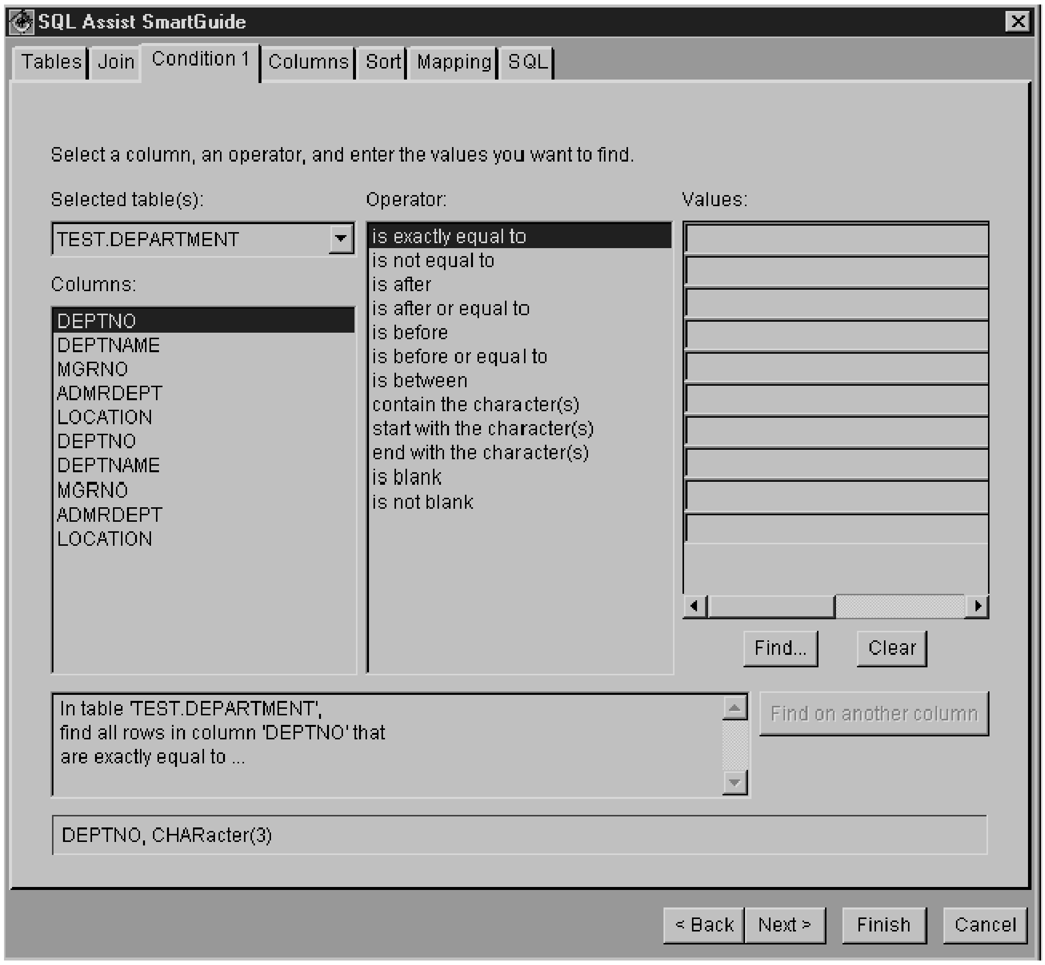
You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

To cancel the visual composition of the SQL statement, select **Cancel**.

Specifying Search Conditions

Use the Condition page of the SQLAssist SmartGuide to specify a search condition for an SQL statement. You can specify multiple search conditions; you specify each search condition in a separately numbered Condition page. The search conditions supplement any joins specified in the Join page, that is, the joins and the search conditions appear in the WHERE clause of the SQL statement.



Select a column, an operator, and enter the values you want to find.

Selected table(s): TEST.DEPARTMENT

Columns:

DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION
DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION

Operator:

is exactly equal to
is not equal to
is after
is after or equal to
is before
is before or equal to
is between
contain the character(s)
start with the character(s)
end with the character(s)
is blank
is not blank

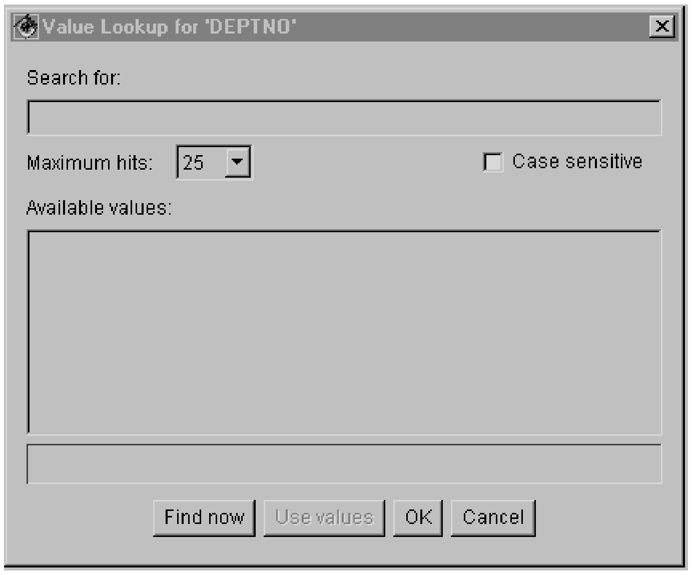
Values:

In table 'TEST.DEPARTMENT',
find all rows in column 'DEPTNO' that
are exactly equal to ...

DEPTNO, CHARacter(3)

Specifying a search condition

1. Select the table for the search from the **Selected table(s)** drop-down list. The list includes only the tables that are selected in the Tables page. The information area displays the current description of the search condition.
2. Select the column for the search from the **Columns** list. Notice that the description of the search condition is updated. Also notice that name and data type of the selected column is displayed.
3. Select an operator from the **Operator** list. The description of the search condition is updated.
4. Specify one or more values in the **Values** list. The description of the search condition is updated. You can enter values. You can also select **Find** to find appropriate values; this opens the Value Lookup window.



Use the Value Lookup window to find appropriate values for a search condition. You can look for values that include a specific character string or you can display all the values in the column. The % character is a wildcard character. For example, specifying A% searches for values that begin with the character A. Specifying %1 searches for values that end with the character 1. The specification A%1 searches for values that begin with the character A and end with the character 1.

Specify the character string in the **Search for** field and select **Find now**. Check the **Case sensitive** checkbox if you want to search for the characters in upper or lower case, exactly as entered in the Search for field.

Select a value from the **Maximum hits** list to control the number of values returned for the search.

To display all the values in the column, select **Find now**; do not specify anything in the **Search for** field.

Results are displayed in the **Available values** list up to a maximum number of rows as specified by the Maximum hits value. Select an appropriate value or values from the list, and select **Use values**. The selected values are added to the Values list in the Condition page.

Select **Close** to close the Value Lookup window.

To cancel value lookup and remove any values selected from value from the Available values list, select **Cancel**.

To remove the values from the Values list in the Condition page, select **Clear**.

You can also specify parameters in the **Values** list. If a parameter is specified, its value is used in the search condition. A parameter is specified in the format :*parm*, where *parm* is the parameter name. For example, :empid is a valid specification for a parameter named empid.

Specifying additional search conditions

After you specify the first search condition, select **Find on another column**. This displays a second search condition window (the tab for the page is labeled Condition 2). Specify the second search condition as described in “Specifying a search condition” on page 24.

Specify a third condition for the query by selecting **Find on another column** in the Condition 2 page. Repeat the process until you specify all the search conditions for the query.

Removing a search condition

Select the condition page for the condition. Then select **Delete Condition**.

When you finish specifying search conditions

When you finish specifying the search conditions, select **Next**. This displays the Columns page of the SQLAssist SmartGuide. Use that page to specify the table columns to display in the result set.

Select **Back** to display the Join page of the SQLAssist SmartGuide. If appropriate, you can then change the joins specified for the SQL statement.

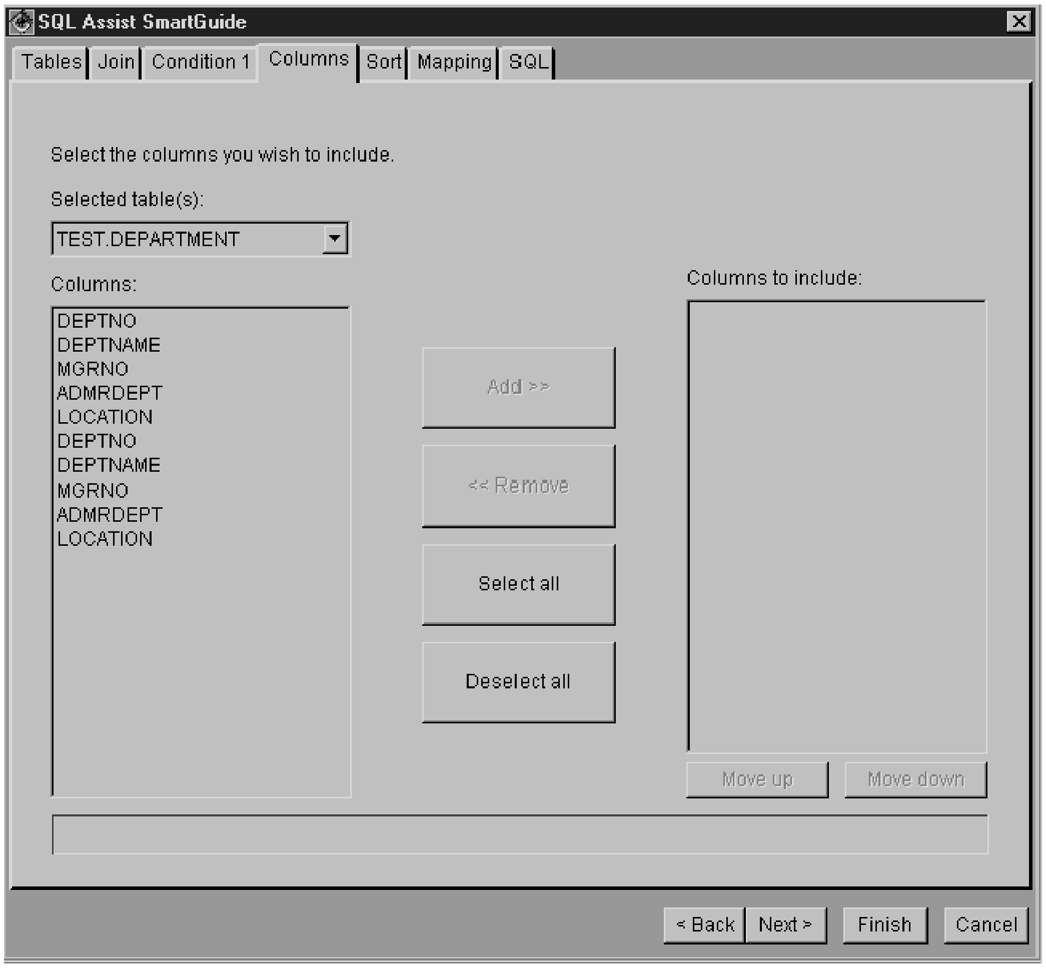
You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

To cancel the visual composition of the SQL statement, select **Cancel**.

Specifying Result Columns

Use the Columns page of the SQLAssist SmartGuide to specify the columns in the result set. The column names will appear in the SELECT clause of the SQL statement.



Select the columns you wish to include.

Selected table(s): TEST.DEPARTMENT

Columns:

DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION
DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION

Columns to include:

1. Select a table from the **Selected table(s)** drop-down list. The list includes only the tables that are selected in the Tables page.
2. Select one or more columns from the **Columns** list. The list includes the columns for the selected table. Use **Select all** to select all the columns in the list. Use **Deselect all** to deselect all selected columns in the list.
3. Select **Add** to add the selected columns to the **Columns to include** list. The selected columns are removed from the **Columns** list.

   To remove one or more columns from the **Columns to include** list, select the columns and select **Remove**. You can navigate through the list by selecting **Move down** or **Move up**. The selected columns are added to the **Columns** list.

When you finish specifying the result columns, select **Next**. This displays the Sort page of the SQLAssist SmartGuide. Use that page to sort the result set.

Select **Back** to display the Condition page of the SQLAssist SmartGuide. If appropriate, you can then change the search conditions for the SQL statement.

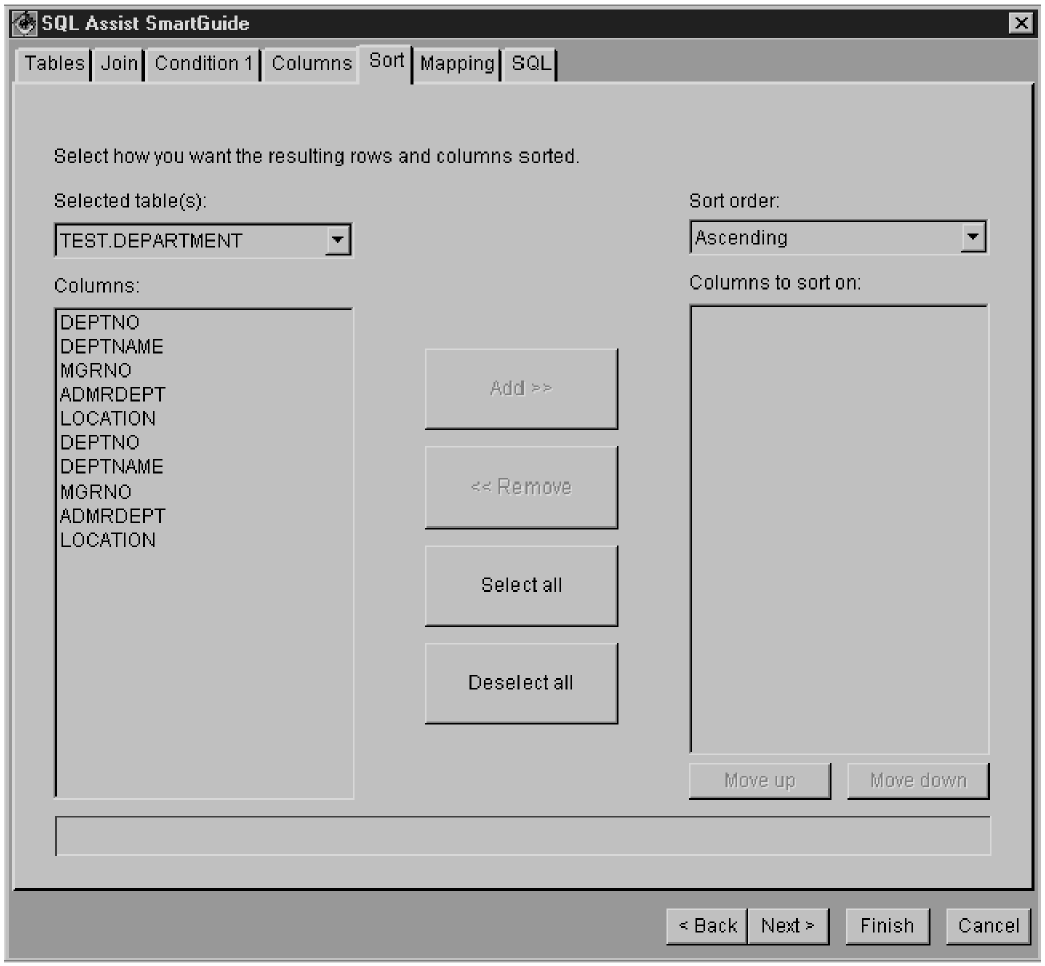
You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

To cancel the visual composition of the SQL statement, select **Cancel**.

Sorting the Result Set

Use the Sort page of the SQLAssist SmartGuide to specify the order of rows in the result set. You specify the order by identifying a column to be used as a sort key. You can specify multiple columns, each one is used as a separate sort key.The rows of the result set are ordered by the value in the selected column, that is by the value of the sort key. If you specify more than one sort key, the rows of the result set are ordered by the value of the first sort key, then by the value of the second sort key, and so on. The sorting specification will appear in the ORDER BY clause of the SQL statement.



Select how you want the resulting rows and columns sorted.

Selected table(s): TEST.DEPARTMENT

Sort order: Ascending

Columns:
DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION
DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION

Columns to sort on:

1. Select a table from the **Selected table(s)** drop-down list. The list includes only the tables that are selected in the Tables page.
2. Select one or more columns from the **Columns** list. The list includes the columns for the selected table. Use **Select all** to select all the columns in the list. Use **Deselect all** to deselect all selected columns in the list.
3. Select **Add** to add the selected columns to the **Columns to sort on** list. The selected columns are removed from the **Columns** list.

   To remove one or more columns from the **Columns to sort on** list, select the columns and select **Remove**. You can navigate through the list by selecting **Move down** or **Move up**. The selected columns are added to the **Columns** list.

4. Select a value from the **Sort order** list. If you select *Ascending*, the result set is ordered in ascending order by values in the selected columns. If you select *Descending*, the result set is ordered in descending order by values in the selected columns.

When you finish specifying the result columns, select **Next**. This displays the Mapping page of the SQLAssist SmartGuide. Use that page to cast the data type of the result to another data type.

Select **Back** to display the Columns page of the SQLAssist SmartGuide. If appropriate, you can then change the columns to be included in the result set.

You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

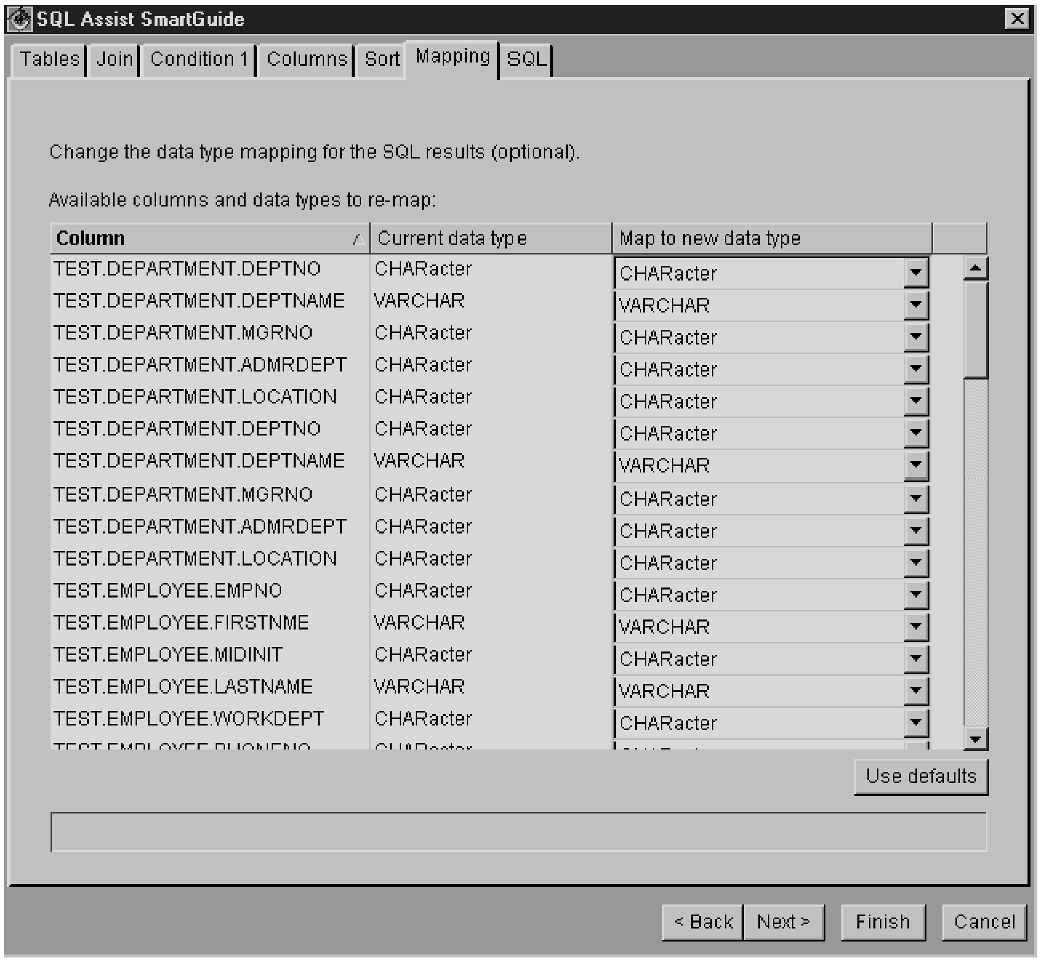
To cancel the visual composition of the SQL statement, select **Cancel**.

Remapping Data to a Different SQL Data Type

By default, the data retrieved by an SQL statement maps to the following Java classes:

SQL Type	**Java class**
CHAR	java.lang.String
VARCHAR	java.lang.String
LONG VARCHAR	java.lang.String
INTEGER	java.lang.Integer
TINYINT	java.lang.Integer
SMALLINT	java.lang.Short
DECIMAL	java.math.BigDecimal
NUMERIC	java.math.BigDecimal
BIT	java.math.Boolean
BIGINT	java.lang.Long
REAL	java.lang.Float
FLOAT	java.lang.Double
DOUBLE	java.lang.Double
BINARY	java.lang.byte[]
VARBINARY	java.lang.byte[]
LONGVARBINARY	java.lang.byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Use the Mapping page of the SQLAssist SmartGuide to remap the data retrieved from a table column to a different SQL data type, and thus, to a different Java class.



Change the data type mapping for the SQL results (optional).

Available columns and data types to re-map:

Column	Current data type	Map to new data type
TEST.DEPARTMENT.DEPTNO	CHARacter	CHARacter
TEST.DEPARTMENT.DEPTNAME	VARCHAR	VARCHAR
TEST.DEPARTMENT.MGRNO	CHARacter	CHARacter
TEST.DEPARTMENT.ADMRDEPT	CHARacter	CHARacter
TEST.DEPARTMENT.LOCATION	CHARacter	CHARacter
TEST.DEPARTMENT.DEPTNO	CHARacter	CHARacter
TEST.DEPARTMENT.DEPTNAME	VARCHAR	VARCHAR
TEST.DEPARTMENT.MGRNO	CHARacter	CHARacter
TEST.DEPARTMENT.ADMRDEPT	CHARacter	CHARacter
TEST.DEPARTMENT.LOCATION	CHARacter	CHARacter
TEST.EMPLOYEE.EMPNO	CHARacter	CHARacter
TEST.EMPLOYEE.FIRSTNME	VARCHAR	VARCHAR
TEST.EMPLOYEE.MIDINIT	CHARacter	CHARacter
TEST.EMPLOYEE.LASTNAME	VARCHAR	VARCHAR
TEST.EMPLOYEE.WORKDEPT	CHARacter	CHARacter
TEST.EMPLOYEE.PHONENO	CHARacter	[illegible]

1. Select column from the **Column** list. **Current data type** displays the SQL data type for the column.
2. Select an SQL data type from the drop-down list in **Map to new data type**. The data retrieved from the column will be mapped to the selected SQL data type.

To reset the mapping of all columns to their default SQL data types, select **Use Default**.

When you finish specifying the result columns, select **Next**. This displays the SQL page of the SQLAssist SmartGuide. Use that page to view the SQL statement. You can also use the page to test or save the SQL statement.

Select **Back** to display the Sort page of the SQLAssist SmartGuide. If appropriate, you can then change the ordering of the result set.

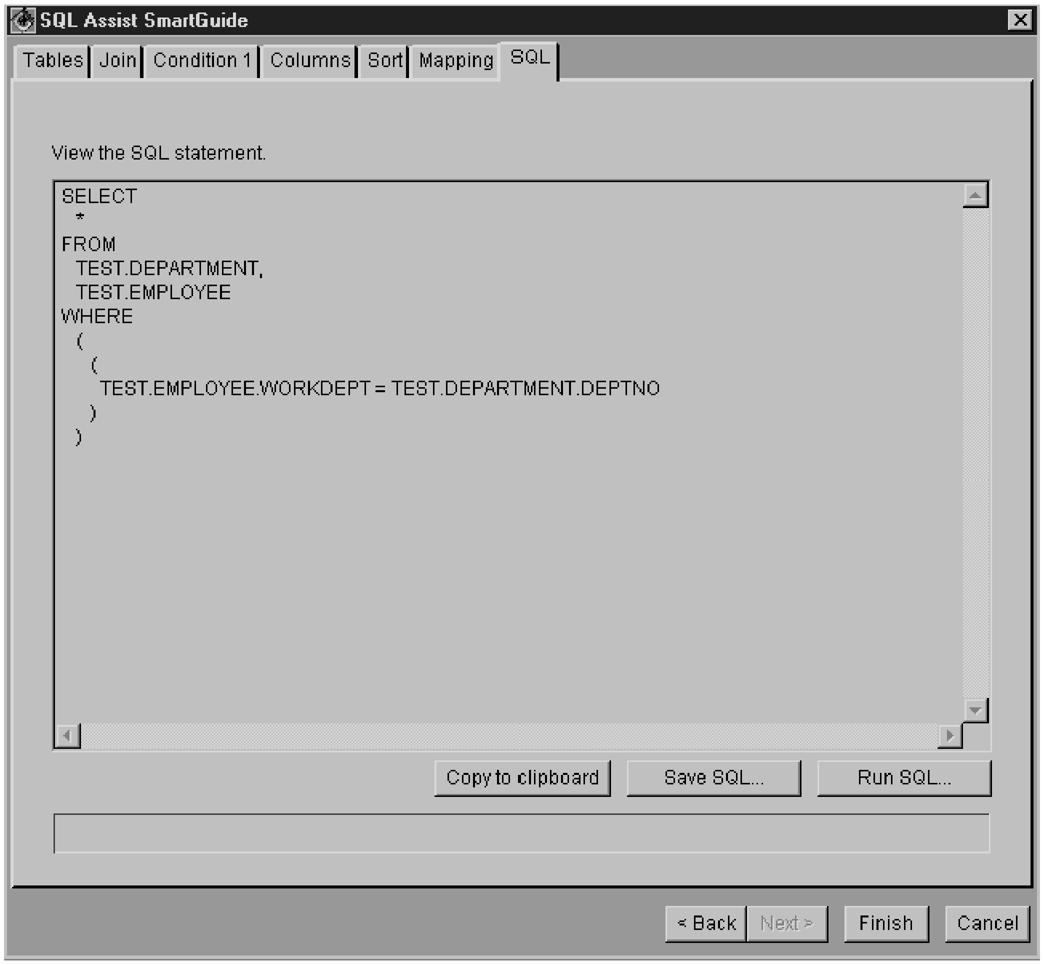
You can also display any page in the SQL Assist SmartGuide by selecting its tab.

When you complete the specification of your SQL statement, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

To cancel the visual composition of the SQL statement, select **Cancel**.

Displaying the SQL statement

Use the SQL page of the SQL Assist SmartGuide to display the SQL statement for a Select bean.



```
SELECT
  *
FROM
  TEST.DEPARTMENT,
  TEST.EMPLOYEE
WHERE
  (
    (
     TEST.EMPLOYEE.WORKDEPT = TEST.DEPARTMENT.DEPTNO
    )
  )
```

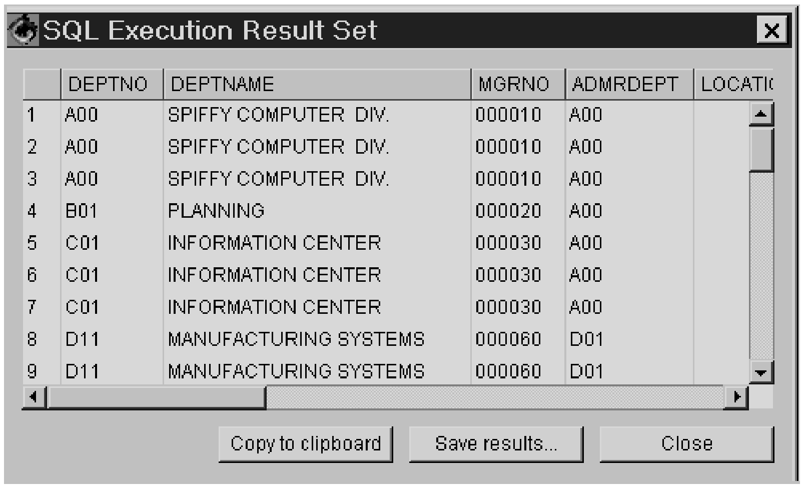
After you display the SQL statement, you can copy it to the clipboard, run it as a test from the SQL page, or save it to a file.

Copying the SQL statement

Select **Copy to clipboard** to copy the SQL statement to the clipboard. The copied SQL statement is available for pasting into other windows such as the SQL Editor window.

Testing the SQL statement

Select **Run SQL** to execute the SQL statement. The SQL statement is run against the database specified in the currently selected connection alias for the Select bean. The result set is displayed in a separate window.


SQL Execution Result Set

	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATI
1	A00	SPIFFY COMPUTER DIV.	000010	A00	
2	A00	SPIFFY COMPUTER DIV.	000010	A00	
3	A00	SPIFFY COMPUTER DIV.	000010	A00	
4	B01	PLANNING	000020	A00	
5	C01	INFORMATION CENTER	000030	A00	
6	C01	INFORMATION CENTER	000030	A00	
7	C01	INFORMATION CENTER	000030	A00	
8	D11	MANUFACTURING SYSTEMS	000060	D01	
9	D11	MANUFACTURING SYSTEMS	000060	D01	

From the SQL Execution Result Set window, you can:

- Copy the result set to the clipboard. Select **Copy to clipboard**.
- Save the result set to a file. Select **Save results**.

Saving the SQL statement

Select **Save SQL** to save the SQL statement in a file. (You can also save the SQL statement from the SQL Execution Result Set window.) You are prompted for a file name and a save location.

You can edit any part of the SQL statement by selecting the appropriate page in the SQL Assist SmartGuide. Selecting **Back** displays the Mapping page. Use this page to remap columns in the result set to different SQL data types.

If the specification of your SQL statement is complete, select **Finish**. This generates the code for the SQL statement and closes the SQLAssist SmartGuide.

To cancel the visual composition of the SQL statement, select **Cancel**.

Executing a Select Bean

To access relational data using a Select bean, you connect an interface component to the Select bean. For example, you can make an event-to-method connection between the *actionPerformed* event for a button and the *execute* method of the Select bean. When the button is selected, the SQL statement associated with the Select bean is executed.

If you have defined parameters in your SQL statement, you must set the parameters before you invoke the *execute* method. If you used the SQL Assist SmartGuide to compose the SQL statement, VisualAge for Java generates two bound properties for each parameter you defined. One property is the parameter in its specified data type. The other property is a String representation of the parameter. So, for example, you can make a property-to-property connection between the text property of a text field and the String representation of a parameter. When you make the connection, code is generated to invoke the *setParameterFromString* method; the method sets the parameter value.

Alternatively, you can connect the DBNavigator bean to the Select bean. The DBNavigator bean provides a set of buttons that includes an Execute button. The DBNavigator bean is a Swing component, and requires the Java Foundation Classes (JFC) library.

To use the DBNavigator bean, you create a property-to-property connection between the *this* property of the Select bean and the *model* property of the DBNavigator bean. The *this* property refers to the whole object of the Select bean. The *model* property specifies that the DBNavigator bean will navigate the associated Select bean. When selected, the Execute button in the DBNavigator bean invokes the *execute* method of the Select bean, which executes the SQL statement.

When you execute an SQL statement using a Select bean, it returns a result set. However the actual number of rows fetched into memory (cache) is controlled by various properties of the Select bean. You can set the value of these properties and thus control how many rows are fetched. The properties determine:

- The maximum number of rows that can be fetched into the cache
- The size of a packet, that is a set of rows
- The maximum number of packets allowed in the cache
- Whether all the rows in the result set are to be fetched into the cache, or only a subset of the result set. “All the rows” means the lesser of:
  - The maximum number of rows
  - The product of the packet size times the maximum number of packets allowed in the cache.

  A “subset of the result set” means only the number of rows needed to satisfy a request. For example, if the result set is 100 rows, but the application only displays 10 rows, only 10 rows are fetched into the cache.

RELATED CONCEPTS

“Chapter 1. About Relational Database Access” on page 1

RELATED TASKS

“Editing Select Bean Properties” on page 8

“Adding the DBNavigator Bean to the Visual Composition Editor Surface” on page 34

RELATED REFERENCES

“Chapter 5. Data Access Beans” on page 39

Displaying and Navigating the Result Set

You can display data in the result set by making a property-to-property connection between the appropriate source property of the Select bean and an appropriate target property of an interface component such as a text field. The Select bean has two bound properties for each data column in the SQL specification. One property is the data column in its specified data type, another is a String representation of the data column. So, for example, you can make a property-to-property connection between the String representation of a data column in the result set and the text property of a text field. When you make the connection, code is generated to invoke the getColumnValue method; the method gets the column value.

Many of the Select bean methods are designed to operate on the current row of the result set. When an SQL statement is executed using a Select bean, the first row of the result set is the current row. If a column value of the result set is connected to an interface component such as a text field, the column value in the current row is displayed in the text field.

You can navigate the result set by using the DBNavigator bean. Using the DBNavigator bean, you can:

- Set the currentRow property of the associated Select bean to the first row in the result set.
- Set the currentRow property of the associated Select bean to the last row in the result set.
- Set the currentRow property of the associated Select bean to the next row in the result set.
- Set the currentRow property of the associated Select bean to the previous row in the result set.

When the current row changes, the values of the bound column properties also change to reflect the values of the new current row. Navigating through the result set changes the value of the data displayed in any interface component connected to the Select bean.

RELATED CONCEPTS

"Chapter 1. About Relational Database Access" on page 1

RELATED TASKS

"Editing Select Bean Properties" on page 8

"Adding the DBNavigator Bean to the Visual Composition Editor Surface"

RELATED REFERENCES

"Chapter 6. Select" on page 41

"Chapter 7. DBNavigator" on page 43

Adding the DBNavigator Bean to the Visual Composition Editor Surface

The DBNavigator bean is a visual bean that you use with a Select bean to access data in a relational database. The DBNavigator bean provides a set of buttons that execute the SQL statement for the associated Select bean; perform other relational database operations, such as commit updates to the database; and navigate rows in the result set.

You add the DBNavigator bean to the Visual Composition Editor surface as follows:

1. From the category drop-down menu in the Visual Composition Editor, select the **Database** category.
2. Select .
3. Move the mouse pointer to the location on the Visual Composition Editor surface where you want to place the DBNavigator bean.

4. Press and hold mouse button 1. Without releasing the mouse button, move the mouse pointer to position it precisely.
5. Release the mouse button. The DBNavigator bean is placed at the location of the mouse pointer.

RELATED REFERENCES

"Chapter 5. Data Access Beans" on page 39

Inserting, Updating, or Deleting Data

The Select bean provides methods that you can use to insert, update, and delete relational data. To perform these operations you must first use a Select bean to retrieve a result set. You then apply the changes to the current row of the result set, as indicated by the *currentRow* property of the Select bean.

Inserting data

To insert data, make an event-to-script connection between an appropriate interface component, such as a button, and a script. For example, you can make an event-to-script connection between the *actionPerformed* event for a button and a script. In the script include code that inserts the row in the result set, use the Select bean method *newRow* to insert the row; then use the *setColumnValue* method to provide the value for the insert. The row is inserted into the database when another row in the result set becomes the current row.

Updating data

To update data, make an event-to-script connection between an appropriate interface component, such as a button, and a script. For example, you can make an event-to-script connection between the *actionPerformed* event for a button and a script. In the script include code that provides an input value for the update, you can use the setColumnValue method to provide the value; use the Select bean method *updateRow* to update the current row in the result set and in the database.

If you made a property-to-property connection between a column property (which is bound) and another property (which might not be bound), VisualAge for Java generates code to propagate the updated value of the bound column property. If the other property is bound, VisualAge for Java generates similar code to propagate the updated value to the column property. If the other property is not bound, you must specify in the connection properties, an event to trigger the propagation of the updated value to the column property. VisualAge for Java then generates code that calls the setColumn method to set the value in the result set when the event is triggered.

Deleting data

To delete data, make an event-to-method connection between an appropriate interface component, to the Select bean. For example, you can make an event-to-script connection between the *actionPerformed* event for a button and the *deleteRow* method of the Select bean. The *deleteRow* method deletes the current row in the result set and in the database.

Using the DBNavigator to insert, delete, or update data

You can use the DBNavigator bean to insert or delete data in a relational database. The DBNavigator bean provides a set of buttons that includes an Insert and Delete button. To use the DBNavigator bean, you create a property-to-property connection between the *this* property of the Select bean and the *model* property of the DBNavigator bean. The *this* property refers to the whole object of the Select bean. The *model* property specifies that the DBNavigator bean will navigate the associated Select bean. When selected, the Insert button in the DBNavigator bean invokes the *newRow* method of the Select bean. When selected, the Delete button in the DBNavigator bean invokes the *deleteRow* method of the Select bean.

You can also use the DBNavigator to update data in a relational database, although there is no Update button provided by the DBNavigator. Change the displayed value, as appropriate, in a interface component, such as a text field, that is connected to the pertinent column value in the result set. If the connection specifies an event to trigger the propagation of the updated value, the value will be set in the result set. Then select a DBNavigator button, such as Next or Last, that changes the currentRow property value. The values of the current row in the result set are updated in the database before the currentRow property value is changed.

RELATED CONCEPTS

"Chapter 1. About Relational Database Access" on page 1

RELATED TASKS

"Editing Select Bean Properties" on page 8

"Adding the DBNavigator Bean to the Visual Composition Editor Surface" on page 34

RELATED REFERENCES

"Chapter 5. Data Access Beans" on page 39

Chapter 4. Data Access Beans

VisualAge provides a set of data access beans that you can use for querying relational databases. The following topics describe these beans:

- “Chapter 5. Data Access Beans” on page 39
- “Chapter 8. Query Property Editor” on page 47
- “New Database Access Class Window” on page 48
- “Connection Alias Definition Window” on page 48
- “New SQL Specification Window” on page 49
- “SQL Editor” on page 50
- “SQL Assist SmartGuide” on page 50
- “Table Name Filter window” on page 54
- “Value Lookup Window” on page 54
- “SQL Execution Result Set Window” on page 55

Chapter 5. Data Access Beans

The following beans provide access to relational data:

Bean	Description
"Chapter 6. Select" on page 41	A non-visual bean used to access relational data
"Chapter 7. DBNavigator" on page 43	A visual bean that provides a set of buttons used with a Select bean

RELATED TASKS

"Chapter 3. Accessing Relational Data" on page 7

Chapter 6. Select

Use a Select bean to access relational data.

Palette category
: **Database**

Palette bean

Project
: IBM Data Access Beans

Package
: com.ibm.ivj.db.uibeans

Type
: Select

The Select bean has the following properties:

beanName
: Specifies the name of the Select bean instance. It must follow standard naming rules for beans. The default name is *Selectn*, where *n* is the number of Select beans with default names; for example, the first default name is *Select1*.

currentRow
: Specifies the current row of the result set. A value of -1 indicates that there is no current row, that is, an SQL statement has not yet been executed or the result set is empty.

currentRowInCache
: Specifies the current row in cache. A value of -1 indicates that there is no current row, that is, an SQL statement has not yet been executed or the result set is empty.

fillCacheOnExecute
: Specifies whether all the rows of the result set are fetched into memory (cache) or only a subset of the result set. A value of *True* means that all the rows of the result set are fetched, up to a maximum number of rows. The maximum number of rows is the **maximumRows** value, or the product of the **packetSize** value multiplied by the **maximumPacketsInCache** value—whichever is smaller. Suppose a result set is 1000 rows, **fillCacheOnExecute** is *True*, **maximumRows** is *100*, **packetSize** is *10*, and **maximumPacketsInCache** is *50*. Executing an SQL statement fetches 100 rows into the cache, that is, the value of **maximumRows**.

  A *False* value means that only the number of rows in the result set needed to satisfy the SQL statement are fetched into the cache. For example, if a result set is 1000 rows, but the application only displays 10 rows, only 10 rows are fetched into the cache.

  The default value is *True*.

lockRows
: Specifies whether a lock is immediately acquired for the row. A value of

True means a lock is immediately acquired for the current row. A *False* value means a lock is not acquired for the row until an update request is issued. The default value is *False*.

maximumPacketsInCache
Specifies the maximum number of packets allowed in the cache. A packet is a set of rows. A value of *0* means that there is no maximum. The default value is *0*.

maximumRows
Specifies the maximum number of rows that can be fetched into the cache. A value of *0* means that there is no maximum. The default value is *0*.

packetSize
Specifies the number of rows in a packet. A value of *0* means that there is no maximum. The default value is *0*.

query Specifies the connection alias and SQL specification for the Select bean. See “Specifying a Connection Alias” on page 9 and “Making an SQL Specification” on page 13 for further information.

readOnly
Specifies whether updates to the data are allowed. A *True* value means that updates are disallowed even if the database manager would permit them. A *False* value means that updates are allowed, provided that the database manager permits them. he default value is *False*.

RELATED TASKS

“Chapter 3. Accessing Relational Data” on page 7

RELATED REFERENCES

“Chapter 7. DBNavigator” on page 43

Chapter 7. DBNavigator

Use the DBNavigator bean with a Select bean to access relational data. The DBNavigator bean provides a set of “Buttons” on page 44 that execute the SQL statement for the associated Select bean; perform other relational database operations, such as commit updates to the database; and navigate rows in the result set. The DBNavigator bean is a Swing component, and requires the Java Foundation Classes (JFC) library.

Palette category
: **Database**

Palette bean

Project
: IBM Data Access Beans

Package
: com.ibm.ivj.db.uibeans

Type
: DBNavigator

Properties

The DBNavigator bean has the following properties:

beanName
: Specifies the name of the DBNavigator bean instance. It must follow standard naming rules for beans. The default name is *DBNavigatorn*, where *n* is the number of DBNavigator beans with default names; for example, the first default name is*DBNavigator1*.

model
: Used to associate the DBNavigator bean with the Select bean. The default is a null value.

showCommit
: Specifies if the Commit button is displayed.

  A value of *True* means that the Commit button is displayed. A *False* value means that the Commit button is not displayed. The default value is *True*.

showDelete
: Specifies if the Delete button is displayed.

  A value of *True* means that the Delete button is displayed. A *False* value means that the Delete button is not displayed. The default value is *True*.

showExecute
: Specifies if the Execute button is displayed.

  A value of *True* means that the Execute button is displayed. A *False* value means that the Execute button is not displayed. The default value is *True*.

showFirst
: Specifies if the First button is displayed.

  A value of *True* means that the First button is displayed. A *False* value means that the First button is not displayed. The default value is *True*.

showInsert
: Specifies if the Insert button is displayed.

A value of *True* means that the Insert button is displayed. A *False* value means that the Insert button is not displayed. The default value is *True*.

showLast
Specifies if the Last button is displayed.

A value of *True* means that the Last button is displayed. A *False* value means that the Last button is not displayed. The default value is *True*.

showNext
Specifies if the Next button is displayed.

A value of *True* means that the Next button is displayed. A *False* value means that the Next button is not displayed. The default value is *True*.

showPrevious
Specifies if the Previous button is displayed.

A value of *True* means that the Previous button is displayed. A *False* value means that the Previous button is not displayed. The default value is *True*.

showRefresh
Specifies if the Refresh button is displayed.

A value of *True* means that the Refresh button is displayed. A *False* value means that the Refresh button is not displayed. The default value is *True*.

showRollback
Specifies if the Rollback button is displayed.

A value of *True* means that the Rollback button is displayed. A *False* value means that the Rollback button is not displayed. The default value is *True*.

toolTipsEnabled
Specifies if tool tips are enabled for the buttons. Tool tips are short descriptions of an interface element, such as a button.

A value of *True* means that tool tips are enabled for the buttons. A *False* value means that tool tips are not enabled for the buttons. The default value is *True*.

Buttons

The buttons that can be displayed with the DBNavigator bean are as follows:

Commit. Commits any uncommitted changes to the database made by the associated Select bean or made by any other Select bean that shares the connection alias with the associated Select bean.

Delete. Deletes the current row of the associated Select bean. If the control that displays the data is connected to the bound column properties of the Select bean, its display changes to reflect the deleted row.

Execute. Connects to the database, if necessary, using the connection specified in the connection alias for the associated Select bean, and executes the SQL statement for the associated Select bean.

First. Sets the currentRow property of the associated Select bean to the first row in the result set. If the control that displays the data is connected to the bound column properties of the Select bean, its displays data from the first row in the result set.

Insert. Inserts a new, blank row in the result set at the position specified by the currentRow property of the associated Select bean. If the control that displays the data is connected to the bound column properties of the Select bean, it displays blanks.

Last. Sets the currentRow property of the associated Select bean to the last row in the result set. If the control that displays the data is connected to the bound column properties of the Select bean, it displays data from the last row in the result set.

Next. Sets the currentRow property of the associated Select bean to the next row in the result set. If the control that displays the data is connected to the bound column properties of the Select bean, it displays data from the next row in the result set.

Previous. Sets the currentRow property of the associated Select bean to the previous row in the result set. If the control that displays the data is connected to the bound column properties of the Select bean, it displays data from the previous row in the result set.

Refresh. Executes the SQL statement for the associated Select bean. It is designed to reexecute an SQL statement that was previously executed. The button does not reconnect to the database. If the SQL statement is changed after its initial invocation, the initial version of the query is executed.

Rollback. Rolls back any uncommitted changes to the database made by the associated Select bean or made by any other Select bean that shares the connection alias with the associated Select bean.

RELATED TASKS

"Chapter 3. Accessing Relational Data" on page 7

RELATED REFERENCES

"Chapter 6. Select" on page 41

Chapter 8. Query Property Editor

Use the Query property editor to specify the value of the query property for a Select bean.

Pages

- "Connection Page"
- "SQL Page"

Connection Page

Use the Connection page of the Query property editor to create a connection alias for a Select bean or to select an existing connection alias.

Fields

- **Database Access Class** names a database access class. The format of the database access class is *packagename.classname*, where *packagename* is the name of the package that contains the database access class, and *classname*, is the name of the class. Select from the list, the database access class that you want to use for the connection alias.
- **Connections** names the connection aliases that are in the selected database access class. Select from the list, the connection alias that you want to use for the Select bean.
- **URL** is a read-only field that shows the URL specified in the definition of the selected connection alias.

Push buttons

- **New** defines a new database access class and adds it to the Database Access Class list.
- **Add** defines a connection alias and adds it to the Connections list.
- **Edit** edits the definition of a connection alias in the Connections list.
- **Remove** removes a connection alias from the Connections list.

SQL Page

Use the SQL page of the Query property editor to create an SQL specification for a Select bean or to select an existing SQL specification.

Fields

- **Database Access Class** names a database access class. The format of the database access class is *packagename.classname*, where *packagename* is the name of the package that contains the database access class, and *classname*, is the name of the class. Select from the list, the database access class that you want to use for the SQL specification.
- **SQL** names the SQL specifications that are in the selected database access class. Select from the list, the SQL specification that you want to use for the Select bean.
- **SQL Statement** is a read-only field that shows the SQL statement for the selected SQL specification

Push buttons

- **New** defines a new database access class and adds it to the Database Access Class list.
- **Add** defines an SQL specification and adds it to the SQL list.
- **Edit** edits an SQL specification.
- **Remove** removes an SQL specification from the SQL list.

RELATED TASKS

"Specifying a Connection Alias" on page 9

"Making an SQL Specification" on page 13

New Database Access Class Window

Use the New Database Access Class window to define a new database access class and add it to the Database Access Class list in the Connection page and the SQL page of the Query property editor.

Fields

- **Package** specifies a package name for the database access class
- **Class name** specifies a class name for the database access class.

Push buttons

- **Browse** opens the Open Package window to select a package from a list of packages in the workspace.

RELATED TASKS

"Defining a Database Access Class" on page 11

Connection Alias Definition Window

Use the Connection Alias Definition window to define a new connection alias or to edit an existing connection alias.

Fields

- **Connection Name** specifies the name for the connection alias. The name must be a valid Java method name.
- **URL** Specifies the URL for the database connection. The URL specification must be in the format jdbc:*subprotocol*:*subname*, where *subprotocol* and *subname* identify the data source for the connection. The value of *subprotocol* depends on the JDBC driver used. For example, for the DB2 application JDBC driver, *subprotocol* is db2; for the Oracle thin driver, *subprotocol* is thin.

  The value of *subname* depends on the *subprotocol* specification; the *subname* value provides information to locate the database. For example, a full URL specification for an application accessing a local database named sample through the DB2 application JDBC driver is:

  ```
  jdbc:db2:sample
  ```

  Here, sample is the *subprotocol* value.

By comparison, a full URL specification for an applet using the Sybase jConnect driver to access a database named sample, on a remote server named myserv, through port number 88 on the internet is:

```
jdbc:sybase:Tds:myserver:88/sample
```

Here, the *subprotocol* value includes the database server name, port number, and database name.

- **JDBC Driver Choices** specifies the JDBC driver class for the database connection. For example, the DB2 application JDBC driver class is COM·ibm·db2·jdbc·app·DB2Driver. Select a JDBC driver class from the drop-down list. If the JDBC driver class you want is not in the drop-down list, Select *Other driver*; you can specify the JDBC driver class in the **JDBC Driver input** field.
- **JDBC Driver input** specifies the JDBC driver class for the database connection. Enter the JDBC driver class in this field if you selected *Other driver* in the **JDBC Driver input** field.
- **Connection Properties** specifies any properties to be passed in the database connection request, other than the user ID and password. Specify the properties in the following format:

  ```
  prop=value;prop=value;...
  ```

  where, *prop* is the name of the property, and *value* is the value of the property. In the following example, three properties are passed:

  ```
  proxy=myserver:88;a=1;b=2
  ```
- **User ID** specifies the user ID to be used in the database connection request (unless the user is prompted for the user ID).
- **Password** specifies the password to be used in the database connection request (unless the user is prompted for the password).

Checkboxes

- **Auto-commit**. If checked, specifies that database updates are automatically committed for each SQL statement. If not checked, specifies that database updates are not automatically committed. This checkbox is checked by default.
- **Prompt for logon ID password before connecting**. If checked, specifies that the user will be prompted for a user ID and password before connection to the database. If not checked, specifies that the user ID and password in the **User ID** and **Password** fields of the database connection alias definition will be used in the database connection request.

Push buttons

- **Test Connection** tests the database connection using the specifications in the connection alias definition.

RELATED TASKS

"Defining or Editing a Connection Alias" on page 11

New SQL Specification Window

Use the New SQL window to create a new SQL specification for a Select bean.

Fields

- **SQL Name** specifies the name for the SQL specification . The name must be a valid Java method name.

Radio Buttons

- **Use SQL Assist SmartGuide** specifies that the SQL Assist SmartGuide will be used to visually compose the SQL statement.
- **Manually write SQL** specifies that the SQL statement will be entered manually in the SQL editor.

RELATED TASKS

"Making a New SQL Specification" on page 14

SQL Editor

Use the SQL Editor to manually enter an SQL statement for a Select bean. You also use the SQL Editor to edit an SQL statement that you previously created using the SQL Editor. (An SQL statement created with the SQL Assist SmartGuide is edited using the SQL Assist SmartGuide.)

RELATED TASKS

"Composing an SQL Statement Manually" on page 16

SQL Assist SmartGuide

Use the SQL Assist SmartGuide to visually compose an SQL statement for a Select bean.

Pages

- "Tables Page"
- "Join Page" on page 51
- "Condition Page" on page 51
- "Columns Page" on page 52
- "Sort Page" on page 52
- "Mapping Page" on page 53
- "SQL Page" on page 53

Tables Page

Use the Tables page of the SQL Assist SmartGuide to specify the tables that are accessed in an SQL statement. The table names that are selected will appear in the FROM clause of the SQL statement.

Fields

- **Table name** lists the tables that are accessible in the database identified by the currently selected connection alias. Checking the checkbox for a table named in the list, selects the table for inclusion in the FROM clause of the SQL statement. More than one table can be selected from the list.

Push buttons

- **View schema(s)** displays a prompt for filtering characters. These characters limit the display to only table names whose schema begins with those characters. The % character is a wildcard character.
- **Filter table(s)** displays a prompt for filtering characters. These characters limit the display to only table names beginning with those characters. The filtering characters are case-sensitive. The % character is a wildcard character.

Join Page

Use the Join page of the SQL Assist SmartGuide to join tables in an SQL statement. The join page displays the columns of each table selected in the Tables page. Select displayed columns to join. You can specify multiple joins.

Push buttons

- **Alias** creates an alias for the selected table and display its columns. The displayed columns of an aliased table are available for joining.
- **Join** joins the selected columns. The join is indicated by a red join line.
- **>** or **<** navigate between multiple joins. The selected join is indicated by a red join line.
- **Unjoin** removes the selected join. The join line is removed.
- **Options** specifies the type of join. The type of joins available are:
  - Inner join. This is a request for rows where the values in the joined columns match.
  - Left outer join. This is a request for an inner join and any additional rows in the left table (as viewed in the Join page) that are not already included in the inner join.
  - Right outer join. This is a request for an inner join and any additional rows in the right table (as viewed in the Join page) that are not already included in the inner join.

Condition Page

Use the Condition 1 page of the SQL Assist SmartGuide to specify a search condition for an SQL statement. You can specify multiple search conditions; you specify each search condition in a separately numbered Condition page. The search conditions supplement any joins specified in the Join page, that is, the joins and the search conditions appear in the WHERE clause of the SQL statement.

Fields

- **Select table(s)** lists the tables that are available for specifying a search condition.
- **Columns** lists the columns in the selected table.
- **Operator** lists the operators that can be specified in the search condition.
- **Values** lists the values for the search condition.

Push buttons

- **Find** opens the Value Lookup window to find values in the selected table column.
- **Find on another column** opens another Condition page (with a tab labeled Condition 2) to specify a second search condition for the SQL statement.

Columns Page

Use the Columns page of the SQL Assist SmartGuide to specify the columns for the result set. The columns will appear in the SELECT clause of the SQL statement.

Fields

- **Select table(s)** lists the tables that are available for specifying the result set.
- **Columns** lists the columns in the selected table.
- **Columns to include** lists the columns that are included in the result set.

Push buttons

- **Add** adds the selected columns to the Columns to include list and removes the selected columns from the Columns list.
- **Remove** removes the selected columns from the Columns to include list and adds the selected columns to the Columns list.
- **Select all** selects all the columns in the list.
- **Deselect all** deselects all the selected columns in the list.
- **Move up** selects the column immediately above the currently selected column in the list.
- **Move down** selects the column immediately below the currently selected column in the list.

Sort Page

Use the Sort page of the SQLAssist SmartGuide to specify the order of rows in the result set. You specify the order by identifying a column to be used as a sort key. You can specify multiple columns, each one is used as a separate sort key. The rows of the result set are ordered by the value in the selected column, that is, by the value of the sort key. If you specify more than one sort key, the rows of the result set are ordered by the value of the first sort key, then by the value of the second sort key, and so on. The sorting specification will appear in the ORDER BY clause of the SQL statement.

Fields

- **Select table(s)** lists the tables that are available for specifying the result set.
- **Columns** lists the columns in the selected table.
- **Columns to sort on** lists the columns that are used as sort keys for ordering the result set.

Push buttons

- **Add** adds the selected columns to the Columns to sort on list and removes the selected columns from the Columns list.
- **Remove** removes the selected columns from the Columns to sort on list and adds the selected columns to the Columns list.
- **Select all** selects all the columns in the list.
- **Deselect all** deselects all the selected columns in the list.
- **Move up** selects the column immediately above the currently selected column in the list.
- **Move down** selects the column immediately below the currently selected column in the list.

Mapping Page

Use the Mapping page of the SQLAssist SmartGuide to remap the data retrieved from a table column to a different SQL data type.

Fields

- **Column** lists the columns that are included in the result set.
- **Current data type** is a read-only field that displays the current SQL data type for the column.
- **Map to new data type** lists the SQL data types to which the column can be mapped

.

Push buttons

- **Use Default** resets the mapping of all columns to their default SQL data types.

SQL Page

Use the SQL page of the SQL Assist SmartGuide to display the SQL statement for a Select bean. After you display the SQL statement, you can copy it to the clipboard, run it as a test from the SQL page, or save it to a file.

Fields

- **View the SQL statement** displays the SQL statement.

.

Push buttons

- **Copy to clipboard** copies the SQL statement to the clipboard.
- **Save SQL** saves the SQL statement in a file.
- **Run SQL** executes the SQL statement against the database specified in the currently selected connection alias for the Select bean.

RELATED TASKS

"Specifying the Tables for an SQL Statement" on page 17

"Joining Tables" on page 20

"Specifying Search Conditions" on page 23

"Specifying Result Columns" on page 26

"Sorting the Result Set" on page 28

"Remapping Data to a Different SQL Data Type" on page 29

"Displaying the SQL statement" on page 31

Schema(s) to View window

Use the Schema(s) to View window to add tables to those listed in the Tables page of the SQL Assist SmartGuide.

Fields

- **Enter the additional schema name(s) to view** specifies a character string to use as a schema name filter. These characters add to the table name list the table names whose schema begins with those characters. For example, if you enter TEST, table names whose schema begins with TEST are added to the list, such as the TEST.EMPLOYEE table. The % character is a wildcard character. Use it to position the schema name filter. For example, specifying %ST requests the addition of all table names whose schema ends with the characters ST. The specification T%ST requests the addition of all tables names that begin with T and end with ST.

Push buttons

- **View all** add the name of all tables in the database to the table name list.

  RELATED TASKS

  "Specifying the Tables for an SQL Statement" on page 17

Table Name Filter window

Use the Table Name Filter window to control the tables that are listed in the Tables page of the SQL Assist SmartGuide.

Fields

- **Enter the table name filter** specifies a character string to use as a table name filter. The filtering characters are case-sensitive. These characters limit the display to only table names beginning with those characters. For example, if you enter EMP, only table names that begin with EMP are listed, such as the EMPLOYEE table. The % character is a wildcard character. Use it to position the filter. For example, specifying %ID requests the display of all table names that end with the characters ID. The specification N%ID requests the display of all tables names that begin with N and end with ID.
- **Table type** specifies the types of tables to be included in the filter.

  RELATED TASKS

  "Specifying the Tables for an SQL Statement" on page 17

Value Lookup Window

Use the Value Lookup window to find appropriate values for a search condition. You do this by finding values that currently exist in the table column to be searched. The table column is selected in the Condition page of the SQL Assist SmartGuide. You can use the Value Lookup window to look for values in the table column that include a specific character string or you can display all the values in the table column.

Fields

- **Search for** identifies a character string to search for in the table column. If nothing is entered in this field, the value search returns all values in the table column. The % character is a wildcard character. For example, specifying A% searches for values that begin with the character A. Specifying %1 searches for values that end with the character 1. The specification A%1 searches for values that begin with the character A and end with the character 1.

- **Maximum hits** specifies the maximum number of values that can be returned in the value search. The default is 25
- **Case sensitive** is a checkbox. If checked, the search is case sensitive; the search for values is based on the upper or lower case of the character string in the Search for field, exactly as entered. If the checkbox is not checked, the search for values in not case sensitive; the search for values must match the character string in the Search for field but can be any combination of upper or lower case.
- **Available values** lists the values in the table column that meet the criteria specified in the Search for field.

Push buttons

- **Find now** searches the specified table column for values that contain the character string specified in the Search for field. If no character string is specified, all values in the column are returned.
- **Use values** adds values selected in the Available values list to the Values list in the Condition page of the SQL Assist SmartGuide.
- **Close** closes the Value Lookup window.

RELATED TASKS

"Specifying Search Conditions" on page 23

SQL Execution Result Set Window

The SQL Execution Result set window displays the result set of an SQL statement run from the SQL page of the SQL Assist SmartGuide. From the window, you can copy the result set to the clipboard or save the result set.

Push buttons

- **Copy to clipboard** copies the result set to the clipboard.
- **Save results** saves the result set in a file.
- **Close** closes the SQL Execution Result Set window.

RELATED TASKS

"Displaying the SQL statement" on page 31