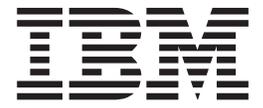IBM VisualAge® for Java™, Version 3.5

Data Access Beans

> **Note!**
>
> Before using this information and the product it supports, be sure to read the general information under **Notices**.

Contents

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.

The Data Access beans are a feature of Visual Age for Java. To use the Data Access beans, you must first add the Data Access Beans feature to your workspace.

Three beans provide core function for accessing databases:
- Select Bean
- Modify Bean
- ProcedureCall Bean

Additional beans provide user interfaces to invoke methods on the core beans and to help display output from the database:
- DBNavigator Bean
- CellSellector Bean
- RowSelector Bean
- ColumnSelector Bean
- CellRangeSellector Bean

All of the beans except for DBNavigator are non-visual.

The Select, Modify, and ProcedureCall beans have properties that contain connection aliases and SQL specifications. These properties allow you to connect to relational databases and access data. You can also use parameterized SQL statements with the Select, Modify, and ProcedureCall beans.

Using Data Access Beans at Run Time

After you develop a program that contains the Data Access beans you can deploy it for use. VisualAge for Java provides several files that are used with these programs 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 a program is deployed.

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

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 programs that use 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

Connection Aliases and SQL Specifications

To connect to a database and access data using a Select, Modify, or ProcedureCall bean, you specify values for the following properties:

- Select bean - query property
- Modify bean - action property
- ProcedureCall bean - procedure property

Each of the properties has the following:

- Connection Alias - the database connection characteristics such as the URL for the connection and the user ID and password passed with the connection request.
- SQL Specification - an SQL statement and meta-data about the statement. VisualAge for Java provides the SQL Assist SmartGuide to help you visually compose an SQL specification or you can use the SQL editor to manually compose an SQL statement. If you manaully compose the SQL statement you may also need to manually add meta-data about it to the generated SQL specification..

Connection aliases and SQL specifications are stored by name in database access classes that VisualAge for Java generates for you. When you create a Select, Modify, or ProcedureCall bean, you can use an exisitng connection alias and/or SQL specification or create new ones. If you create a new one, you can store it in an existing database access class or create a new database access class. Your bean can use a connection alias from one database access class and an SQL specification from another database access class or it can use one database access class for both.

If you create multiple Select, Modify, or ProcedureCall beans for your program, the beans can share one connection alias and/or one SQL specification. If multiple beans use the same connection alias, at run time they share the same physical connection to the database and when one of the beans commits an update to the database, all uncommitted updates made by the other beans are committed also.

Parameterized SQL Statements

The Select, Modify and ProcedureCall beans support the use of parameterized SQL statements. A parameterized SQL statement contains one or more parameters or variables whose value can be changed as your program runs. For example, your SQL statement might be "SELECT * FROM STAFF WHERE DEPT = :deptNo" Your program can change the value for :deptNo each time the statement runs.

When you call a stored procedure, you can use parameters to supply input values. If the stored procedure returns output values (instead of or in addition to result sets), you must use parameters to obtain the output values. For example, your SQL statement might be "CALL MYPROC (5, :in1, :out1)". Your program will always use "5" as the first input value and can change the input value for ":in1" each time the statement runs. It must get the output value of ":out1" from the stored procedure after each run.

The Select, Modify, and ProcedureCall beans have methods to set and get the values of parameters. Each parameter for these beans has a name and can be accessed by name or number. For example, the statement above, "CALL MYPROC (5, :in1, :out1)" has two paraneters. (Because your program cannot change the value 5, it is not a statement parameter.) The last parameter can be identified either with the name "out1" or the number 2.

When you use the SQL Assist SmartGuide to compose your SQL statement, VisualAge for Java generates two bound properties for each parameter; the parameter in its specified data type and a string representation of the parameter. In the Visual Composition Editor you can connect these properties to properties of interface components to get and set parameter values. For example, a property-to-property connection between the string representation of an output parameter and the text property of a text field results in the value displayed in the text field changing whenever the parameter value changes.

RELATED CONCEPTS
Connection Aliases and SQL Specifications
Select Bean
Modify Bean
ProcedureCall Bean

RELATED TASKS
Accessing Relational Data

Select Bean

The Select bean is a non-visual bean. Using the Select bean you can query a relational database. You can also use the result set that is returned to insert, update or delete a row the database.

The Select bean has a query property which specifies how to connect to a database and defines an SQL statement. Other properties allow you to specify such things as when rows are fetched, how many at a time, and whether a database lock should be acquired and held for a row while it is the current row.

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 specified in the query property executes.

When you execute an SQL statement using a Select bean, it returns a result set. Unlike the native Java interface to relational data (JDBC), the Select bean maintains

rows of the result set in a memory cache where you can move back and forth among the rows. You can control how many rows are in the cache at one time or over time by setting Select bean properties that control memory management (page 4).

When an SQL statement runs using a Select bean, the first row of the result set becomes the current row. If you move to another row, that row becomes the current row. Many of the Select bean methods operate on the current row of the result set; other methods allow you to move among the rows.

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 JTable or text field. For example, to display result set data in a tabular form, you can make a property-to property connection between the this property of the Select bean and the model property of the JTable.

You can also display result set data one row at a time. If you use the the SQL Assist SmartGuide to compose your SQL statement, VisualAge for Java will generate two bound properties for each data column in the result set. One property is the data column in its specified data type, the other is a string representation of the data column. Make a property-to-property connection between the String representation of each data column in the result set and the text property of a text field. The text fields will display the column values of the current row.

Inserting, updating and 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 and position to the desired row. Any changes made to rows in the cache are applied to the database as well:

- You can set new values for columns in the current row. When you leave the row, the changes are automatically applied to the database.
- If you delete the current row, it is immediately deleted from the database.
- You can insert a row before or after the current row. Values you set in the new row are automatically inserted into the database when you leave the new row.

There are various ways to use the methods for inserting, updating and deleting data. For example, one way to implement updates in a program is to make an event-to-code connection between an appropriate interface component, such as a button, and a method. This method would obtain the new value from an interface component such as a text field and then use a Select bean method to set the value of the column in the current row of the result set to this new value. The method would then use another Select bean method to update the row in the database with these values.

Memory Management

You can control how many rows are in the memory cache at one time or over time by setting expert properties of the Select bean. The properties determine:

- The maximum number of rows that can be fetched into the cache over time.
- The minimum number of rows to be fetched at one time. This is called packet size. (Rows are fetched into or displaced from the cache in packets.
- The maximum number of packets allowed in the cache at one time. (Older packets may be displaced as newer packets are fetched.)

- Whether the cache should be filled with as many rows as possible (given the settings for the properties above) as soon as you run your query or have rows added as you ask for them.

The maximum number of rows that will be in the cache at one time is the least of:

- The packet size times the maximum number of packets allowed in the cache.
- The maximum number of rows that can be fetched over time.
- All of the rows in the result set.

If memory management is not critical to your program, you need not be concerned with these properties. By default, the cache size is not limited; rows are fetched one at a time (a packet size of 1); and all of the rows in the result set are fetched into the cache as soon as you run your query.

RELATED CONCEPTS
Connection Aliases and SQL Specifications
Parameterized SQL Statements
Modify Bean
ProcedureCall Bean
Selector Beans
Navigator Bean

RELATED TASKS
Accessing Relational Data

Modify Bean

The Modify bean is a non-visual bean. Using the Modify bean you can insert, update, delete rows in the database without first running a query.

The Modify bean has an action property which specifies how to connect to a database and defines an SQL statement. Other properties allow you to get information such as whether the SQL statement has been executed and how many rows in the database were affected by it.

The Modify bean provides an execute method that executes the SQL statement and methods to commit or roll back the changes to the database. To use the Modify bean, you connect an interface component to it. For example, you can make an event-to-method connection between the actionPerformed event for a button and the execute method of the Modify bean. When the button is selected, the SQL statement specified in the action property is executed.

The Modify bean is intended to run SQL INSERT, UPDATE and DELETE statements. However, it can run any valid SQL statement, but it will not return a result set. Use a Select or ProcedureCall bean to obtain result sets. Because the Modify bean doesn't return a result set, you cannot use the DBNavigator bean with it.

RELATED CONCEPTS
Connection Aliases and SQL Specifications
Parameterized SQL Statements
Select Bean
ProcedureCall Bean

ProcedureCall Bean

The ProcedureCall bean is a non-visual bean. Using the ProcedureCall bean, you can run a stored procedure. You can pass parameters as input, output or both and you can access any result sets the stored procedure returns. You can also use any result sets that are returned to insert, update, or delete rows in the database.

The ProcedureCall bean has a procedure property which specifies how to connect to a database and defines an SQL statement. All of the properties you can specify for a Select bean can also be specified for a ProcedureCall bean. In addition, the ProcedureCall bean has properties that govern the handling of any result sets returned by a stored procedure.

The ProcedureCall bean provides a set of methods for running stored procedures and obtaining output from them. For example, the execute method executes the SQL CALL statement. There are methods for setting input parameter values and getting output parameter values. To run a stored procedure, you connect an interface component to the ProcedureCall bean. For example, you can make an event-to-method connection between the actionPerformed event for a button and the execute method of the ProcedureCall bean. When the button is selected, the SQL CALL statement specified in the procedure property executes.

When you run a stored procedure using the ProcedureCall bean, it may return zero, one, or multiple result sets. Unlike the native Java interface to relational data (JDBC), the ProcedureCall bean maintains result sets in a memory cache where you can move back and forth between the result sets and the rows within the result sets. You can control the number of result sets in the cache at one time and the number of rows within each result set at one time or over time by setting ProcedureCall properties that control memory management. (page 4)

Once you have used a Procedure Call bean to run a stored procedure, if it returned one or more result sets, you are always positioned on a result set, referred to as the "current result set". If the current result set has one or more rows, you are also positioned on a row, referred to as the "current row". Many of the Procedure Call bean properties operate on the current row of the current result set. Initially the current result set is the first one returned, and the current row is its first row. The Procedure Call bean includes methods to move to a different row or a different result set. For example, there are methods that move to the next row, the previous row, the next result set, and the previous result set.

Values for parameters used as input to or output from the stored procedure can be accessed via an interface component such as a text field using the bound parameter properties of the Procedure Call bean that Visual Age for Java generates.

If the stored procedure returns one or more result sets, you can display data from the current result set by making a property-to property connection between the appropriate source property of the ProcedureCall bean and the appropriate target property of an interface component such as a JTable or a text field. To display result set data in tabular form make a property-to-property connection between the this property of the ProcedureCall bean and the model property of the JTable. As you move from one result set to another, the table will change accordingly.

You can also display result set data one row at a time. If the stored procedure returns only one result set or all returned result sets have similar columns and you use the SQL Assist SmartGuide to describe a result set, Visual Age for Java will generate two bound properties for each data column in the result set. One property is the data column in its specified data type, the other is a string representation of the data column. Make a property-to-property connection between the string representation of each data column in the result set and the text property of a text field, and the text field displays the column values of the current row of the current result set.

Inserting, updating and deleting data

The ProcedureCall bean provides methods that you can use to insert, update, and delete relational data. To perform these operations, the ProcedureCall bean must return one or more result sets and be positioned to the desired result set and the desired row. Any changes made to rows in the cache are applied to the database as well:

- You can set new values for columns in the current row. When you leave the row, the changes are automatically applied to the database.
- If you delete the current row, it is immediately deleted from the database.
- You can insert a row before or after the current row. Values you set in the new row are automatically inserted into the database when you leave the new row.

There are various ways to use the methods for inserting, updating and deleting data. For example, one way to implement updates in a program is to make an event-to-code connection between an appropriate interface component, such as a button, and a method. This method would obtain the new value from an interface component such as a text field and then use a ProcedureCall bean method to set the value of the column in the current row of the current result set to this new value. The method would then use another ProcedureCall bean method to update the row in the database with these values.

Memory Management

By setting expert properties of the ProcedureCall bean, you can control how many result sets are in the memory cache at one time, and for each result set, how many rows are in the memory cache at one time or over time.

The properties for result sets determine:
- The maximum number of result sets allowed in the cache at one time. (Older result sets may be displaced as newer ones are fetched.
- Whether the cache should be filled with as many result sets as possible (given your setting for the property above) as soon as the stored procedure runs or should have result sets fetched as you ask for them.

The properties for rows determine:
- The maximum number of rows that can be fetched into the cache over time for each result set.
- The minimum number of rows to be fetched at one time. This is called packet size. (Rows are fetched into or displaced from the cache in packets.
- The maximum number of packets allowed in the cache at one time for each result set. (Older packets may be displaced as newer packets are fetched.)
- Whether the cache should be filled with as many rows as possible for each result set (given the settings for the properties above) as soon as you run your the stored procedure or have rows added as you ask for them.

The maximum number of rows that will be in the cache at one time is the least of:

- The packet size times the maximum number of packets allowed in the cache.
- The maximum number of rows that can be fetched over time.
- All of the rows in the result set.

If memory management is not critical to your program, you need not be concerned with these properties. By default, the cache size is not limited; rows are fetched one at a time (a packet size of 1); and all of the rows of all result sets are fetched into the cache as soon as you run your store procedure.

Only the last result set fetched remains open to fetch additional rows. Therefore, if you fetch all of the result sets as soon as the stored procedure runs, you probably want to fetch all of the rows for each result set at that time. Otherwise, there may be rows in result sets, other than the last one, which cannot be fetched.

RELATED CONCEPTS
Connection Aliases and SQL Specifications
Parameterized SQL Statements
Select Bean
Modify Bean
Selector Beans
Navigator Bean

RELATED TASKS
Accessing Relational Data

DBNavigator Bean

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

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 or ProcedureCall beans and the model property of the DBNavigator bean. The this property refers to the whole object of the non-visual bean. The model property specifies which non-visual bean the DBNavigator bean will navigate.

RELATED CONCEPTS
Select Bean
ProcedureCall Bean

RELATED TASKS
Accessing Relational Data

Selector Beans

Selector beans are non-visual beans. They provide views of subsets of the data in a result set. There are four Selector beans:

- CellSelector - provides a single value view of data.
- ColumnSelector - provides a column view of data.
- RowSelector - provides a row view of data.
- CellRangeSelector - provides a two-dimensional view of data.

All Selector beans will work with data presented in any implementation of the Java TableModel interface such as the Select and ProcedureCall bean. All Selector beans have properties you can use to define the subset of tabular data you wish to work with. These properties can be modified at run-time in order to navigate through the data.

The ColumnSelector and RowSelector implement the Java ComboBoxModel interface and the CellRangeSelector implements the Java TableModel interface. This allows the selectors to act as the model for other Java classes that use these interfaces such as a JList or a JTable.

In addition, all Selector beans have data access properties that allow access to their source data converted to a specified data type. There are properties for each of the base Java data types and several common Java classes to support text strings, dates, and times. These properties allow other beans to see the source data in a form they understand, avoiding the need for them to use the Java TableModel interface.

RELATED CONCEPTS
Select Bean
ProcedureCall Bean

RELATED TASKS
Using Selector Beans
Accessing Relational Data

About the Create Database Application SmartGuide

The Create Database Application SmartGuide assists you in creating a database application using the Select bean for data access. Once you define the query property of the Select bean, the Create Database Application SmartGuide generates a graphical user interface (GUI) to display the columns of the result returned from a query.

In particular, you use the Create Database Application SmartGuide to create the following components of a database application:

- A database connection
- An SQL statement
- A graphical user interface

The Create Database Application SmartGuide is installed when you add the Data Access beans as a feature of Visual Age for Java. To use the Data Access beans, you must first add the Data Access Beans feature to your workspace. To add the Data Access beans, select **File > Quick Start > Features > Add Features**; then select **Data Access Beans 3.0** and click **OK**.

Starting the Create Database Application SmartGuide
Creating a database application with the Create Database Application SmartGuide

Chapter 2. 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 any of three Data Access beans to access relational data in different ways:

- The Select bean allows you to run a query and access the result set returned by the query. You can insert, update, and delete rows in the result set without writing separate SQL INSERT, UPDATE and DELETE statements. Using the Select bean in the Visual Composition Editor may involve:

  - "Adding a Select, Modify, or ProcedureCall bean to the Visual Composition Editor surface" on page 12
  - "Editing Select bean properties" on page 39
  - "Executing a Select bean" on page 41
  - "Displaying and navigating a result set" on page 55
  - "Inserting, updating, or deleting data in a result set" on page 66

- The Modify bean allows you to run SQL INSERT, UPDATE, or DELETE statements without first running a query and retrieving its result set. Using a Modify bean in the Visual Composition Editor may involve:

  - Adding a Select, Modify, or ProcedureCall Bean to the Visual Composition Editor Surface
  - Editing Modify Bean Properties
  - Executing a Modify Bean

- The ProcedureCall bean allows you to run a stored procedure passing values for input parameters and receiving values for output parameters. If the stored procedure returns one or more result sets, the ProcedureCall bean gives you access to them and you can insert, update or delete rows within the result sets without writing separate SQL INSERT, UPDATE and DELETE statements. Using the ProcedureCall bean in the Visual Composition Editor may involve:

  - Adding a Select, Modify, or ProcedureCall Bean to the Visual Composition Editor Surface
  - Editing ProcedureCall Bean Properties
  - Executing a ProcedureCall Bean
  - Displaying and Navigating a Result Set
  - Inserting, Updating, or Deleting Data in a Result Set

Selector beans allow you to work with a subset of the result set returned by a Select or ProcedureCall bean. As appropriate, the selector beans implement either

the Java ComboBoxModel or TableModel interface, so they can pass data directly to user interfaces such as a JList or JTable. Using Selector Beans in the Visual Composition Editor may involve:

- Adding Seletor Beans to the Visual Composition Editor Surface
- Using Selector Beans

RELATED TASKS
Using the Quick Start Window
Setting the Class Path

Adding a Select, Modify, or ProcedureCall bean to the Visual Composition Editor surface

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

These are nonvisual beans that you use to access data in a relational database. Start by adding one of the beans 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 [icon] for a Select bean, [icon] for a Modify bean, or [icon] for a ProcedureCall bean..

3. Move the mouse pointer to the location on the Visual Composition Editor surface where you want to place the 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 bean is placed at the location of the mouse pointer.

RELATED CONCEPTS
Beans Palette

RELATED TASKS
Adding a feature to VisualAge for Java
Composing beans visually

RELATED REFERENCES
Visual Composition Editor
"Select (Database)" on page 75
Modify
ProcedureCall

Making an SQL specification

You use a Select, Modify, or ProcedureCall bean to access relational data. Use the SQL page of the data property editor to specify the SQL statement for the 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 use the same SQL specification. The same is true for Modify and ProcedureCall beans.
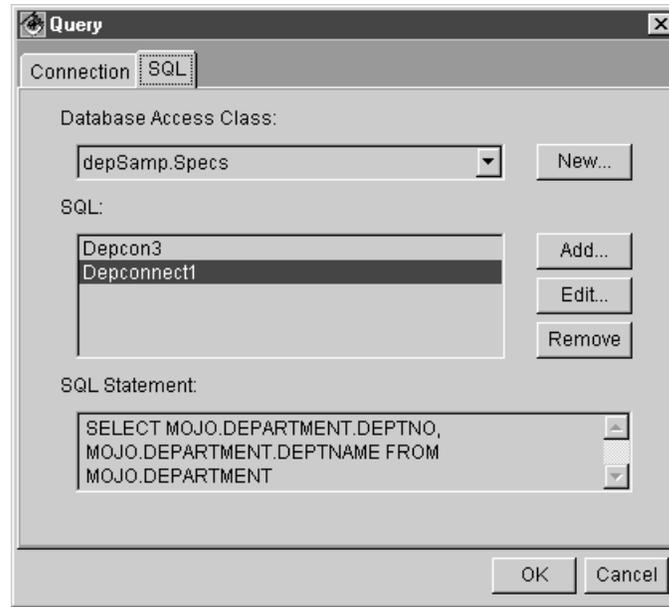
1. Select a database access class from the drop-down list in the **Database Access Class** field. The list shows the database access classes that exist in the workspace. The format of each item in the list is *packagename.classname*, where *packagename* is the name of the package that contains the database 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. If you are editing a Query property, only specifications of queries are shown, likewise for Procedure properties and Action properties.

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

    To edit an SQL specification, select the SQL specification 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

Making a new SQL specification

Select **Add** in the SQL page of the data property editor to make a new SQL specification for a Select, Modify, or ProcedureCall bean. This opens the New SQL Specification window.
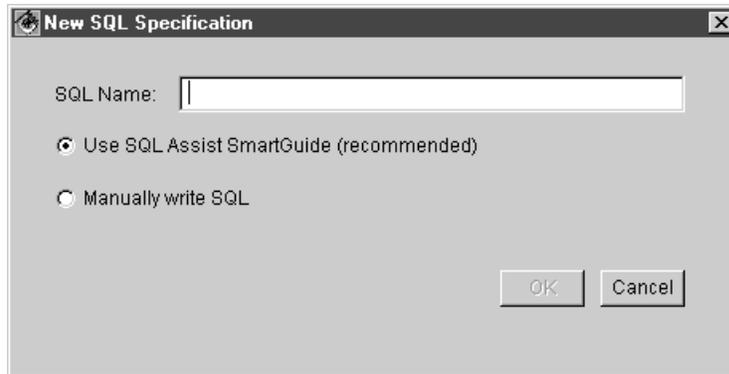
- Specify a name for the SQL specification in the **SQL Name** field. The name must be a valid Java method name. Although only SQL specification definitions of the same type as this one appear in the list on the SQL page of the property editor, the name you provide here must be unique within all of the definitions in the database access class, including SQL specification definitions of other types as well as connection alias definitions.
- 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** radio button 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 specification" on page 15

Defining a database access class

Select **New** in the Connection page of the data property editor to define a new database access class to contain a connection alias for a Select, Modify, or ProcedureCall bean. Select **New** in the SQL page of the data property editor to define a new database access class to contain an SQL specification for a Select, Modify, or ProcedureCall bean. Selecting **New** opens the New Database Access Class window.
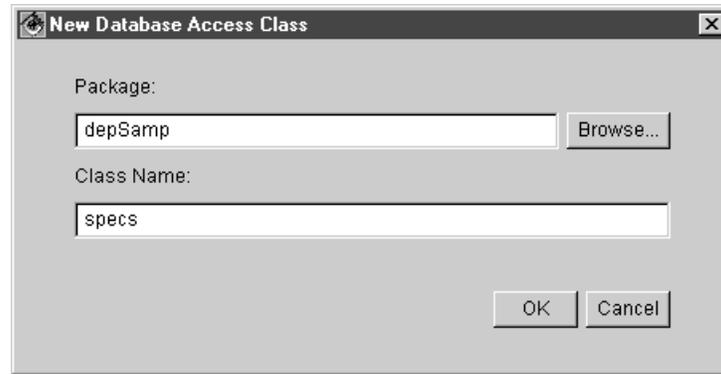
1. Specify a package for the database access class in the **Package** field. Select **Browse** to view a list of the packages available in the workspace.
2. Specify a class name for the database access class in the **Class name** field.
3. When you finish specifying a package and class name, select **OK**. 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 data property editor.

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

Composing or editing an SQL specification

An SQL specification contains an SQL statement plus meta-data about the statement. You can compose a new SQL specification or edit one that you already composed.

You compose a new SQL statement by selecting **Add** in the SQL page of the data 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 and its meta-data visually.
- Use an SQL editor to enter the SQL statement manually. If you choose this option, you may need to add meta-data information to the SQL specification generated in your Database Access class.

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 a data 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 query visually"
Composing an SQL INSERT, UPDATE, or DELETE visually
Composing an SQL procedure call visually
"Composing an SQL statement manually" on page 38
Making a new SQL specification

Composing an SQL query visually

Use the SQL Assist SmartGuide to visually compose an SQL query for a Select bean.
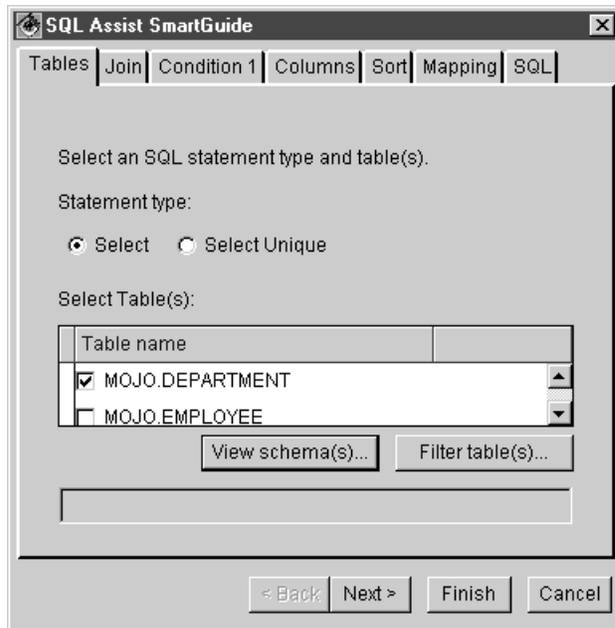
Using the SQL Assist SmartGuide you can:
- Specify the tables to be accessed in the SQL query. (Required.)
- Join the tables. (Optional.)
- Specify search conditions for the SQL query. (Optional.)
- Specify the table columns to return in the result set. (Optional.)
- Sort the result set. (Optional.)
- Remap a result column to a different Java class. (Optional.)
- Display the resulting SQL query. (Optional.)

You must first chose the statement type. Chose the **Select Unique** button if you want to eliminate any duplicate rows in the result set. Chose the **Select** button if you want the result set to contain all of the rows that meet your selection criteria, including any duplicate rows.

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

RELATED TASKS
"Specifying the tables for an SQL statement"

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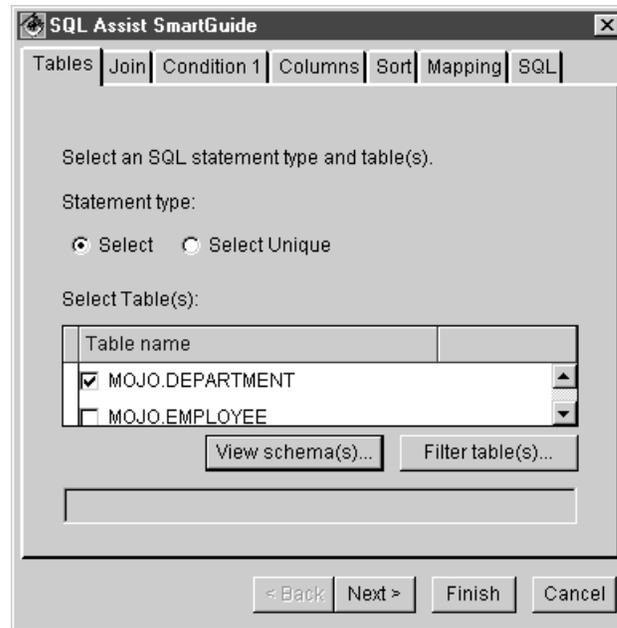
The **Table name** field lists the tables that you can access in the database identified by the currently selected connection alias. By default, the tables listed are those whose schema is the user ID specified for the database connection. If no tables in the database have that schema, all the tables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

You can add or remove table names 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.
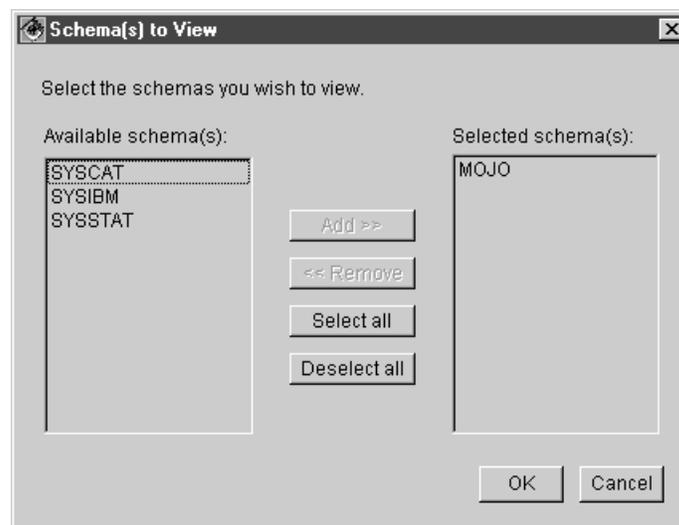
- You can select schemas from Available schema(s) and add them to Selected schema(s) by clicking the **Add** button.
- You can remove schemas from Selected schema(s) using the **Remove** button.
- You can select or deselect all the schemas in both Available schema(s) and Selected schema(s) using the **Select all** or **Deselect all** buttons.

When you finish selecting schemas to view, select **OK**. This closes the Schema(s) to View window.

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

```
┌─────────────────────────────────────────────────┐
│ 🖼 Table Name Filter                          ✕  │
├─────────────────────────────────────────────────┤
│ Enter the table name filter to use below:         │
│ ┌─────────────────────────────────────────────┐ │
│ │ %                                            │ │
│ └─────────────────────────────────────────────┘ │
│                                                   │
│  ┌──────────────────────────────────┬─────────┐ │
│  │ Table type                        │         │ │
│  ├──────────────────────────────────┼─────────┤ │
│  │ ☑  ALIAS                          │         │ │
│  │ ☑  SYSTEM TABLE                   │         │ │
│  │ ☑  TABLE                          │         │ │
│  │ ☑  VIEW                           │         │ │
│  └──────────────────────────────────┴─────────┘ │
│                         ┌──────┐  ┌────────┐     │
│                         │  OK  │  │ Cancel │     │
│                         └──────┘  └────────┘     │
└─────────────────────────────────────────────────┘
```

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 tables, or views by checking the corresponding checkbox in the **Table type** field. You can check multiple table types.

When you finish entering information that filters table names in the list, select **OK**. This closes the Table Name Filter window.

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**.

RELATED TASKS
Joining tables
Specifying search conditions
Specifying result columns
Sorting the result set
Remapping data to a different SQL data type
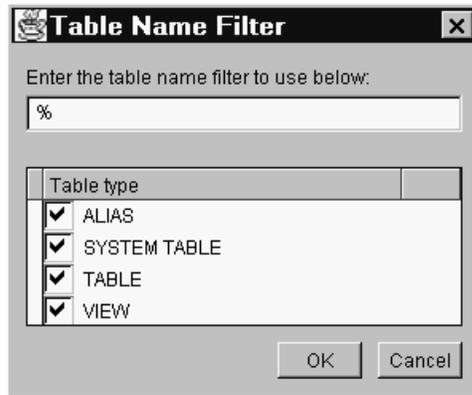Displaying the SQL statement

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.
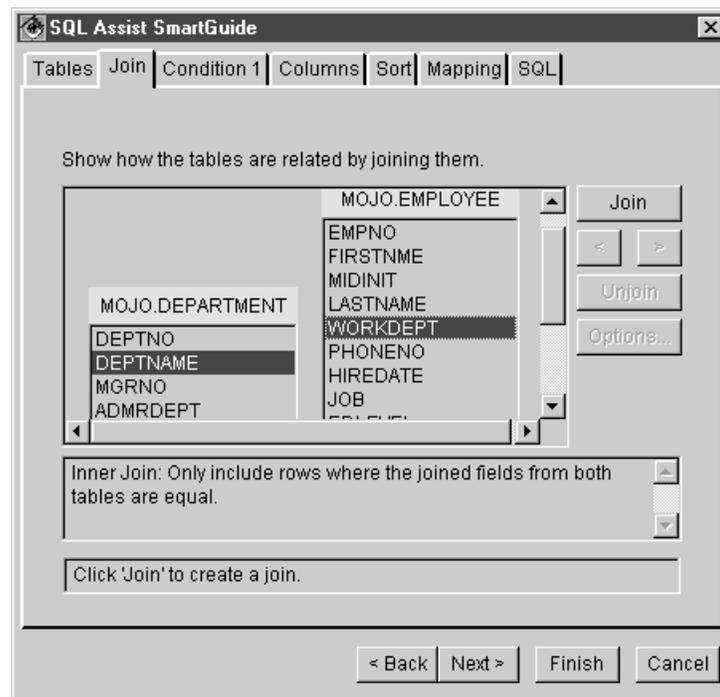
Requesting a join

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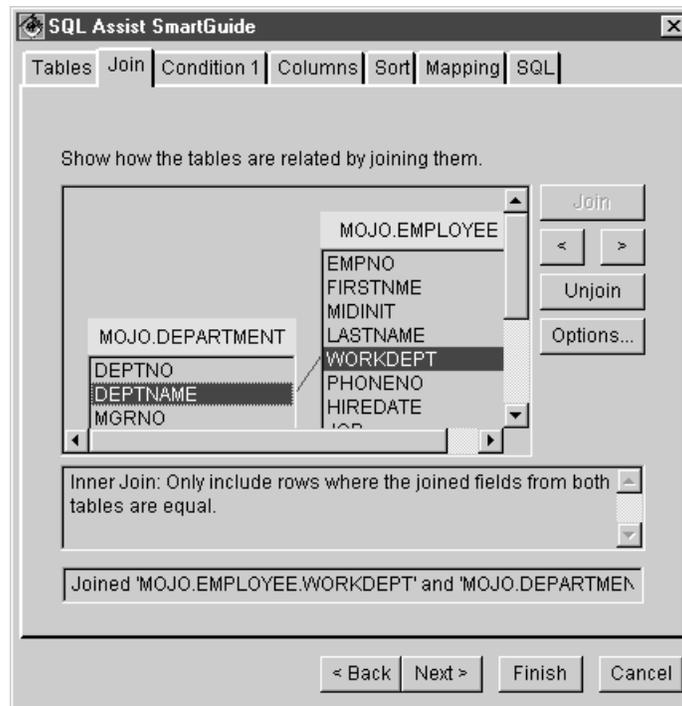
By default, a join request is assumed to be an inner join. An inner join requests only the rows where the values of the two columns match. You can also request other types of joins by selecting **Options**. You can select:

- 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.

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.

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 the tables for an SQL statement
Specifying search conditions
Specifying result columns
Sorting the result set
Remapping data to a different SQL data type
Displaying the SQL statement

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.
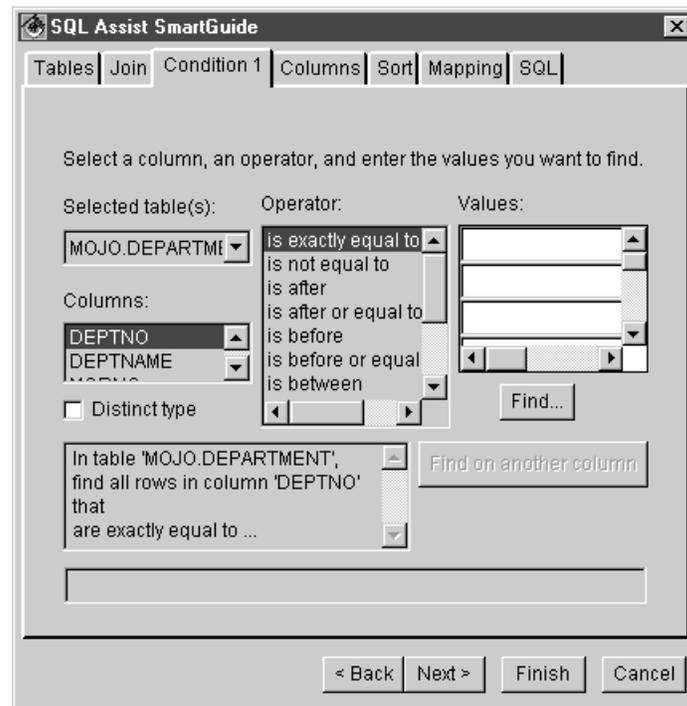
Specifying a search condition

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. Click the **Distinct type** checkbox if the condition involves a column with a user-defined (distinct) type.

3. 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.

4. Select an operator from the **Operator** list. The description of the search condition is updated.

5. 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.
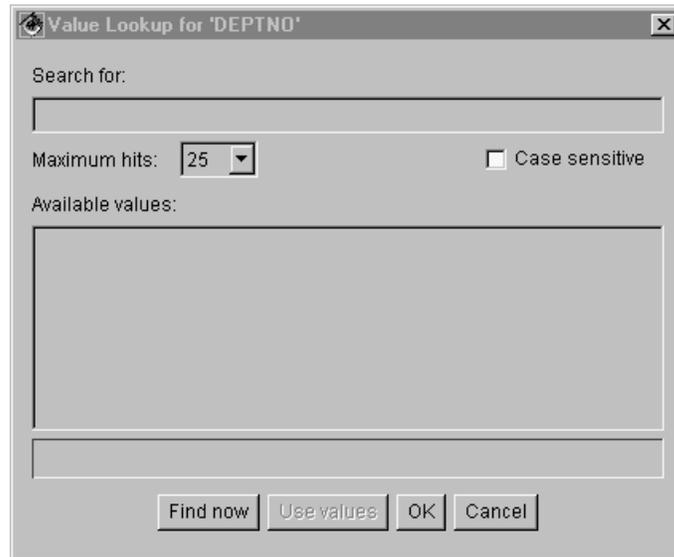
Use the Value Lookup window to find appropriate values for a search condition. You can look for values that include a specific character string or you can display all the values in the column. The % character is a wildcard character. For example, specifying A% searches for values that begin with the 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 window is labeled Condition 2). Specify the second search condition as described in Specifying a search condition (page 21).

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 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**.

RELATED TASKS
Specifying the tables for an SQL statement
Joining tables
Specifying result columns
Sorting the result set
Remapping data to a different SQL data type
Displaying the SQL statement

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.
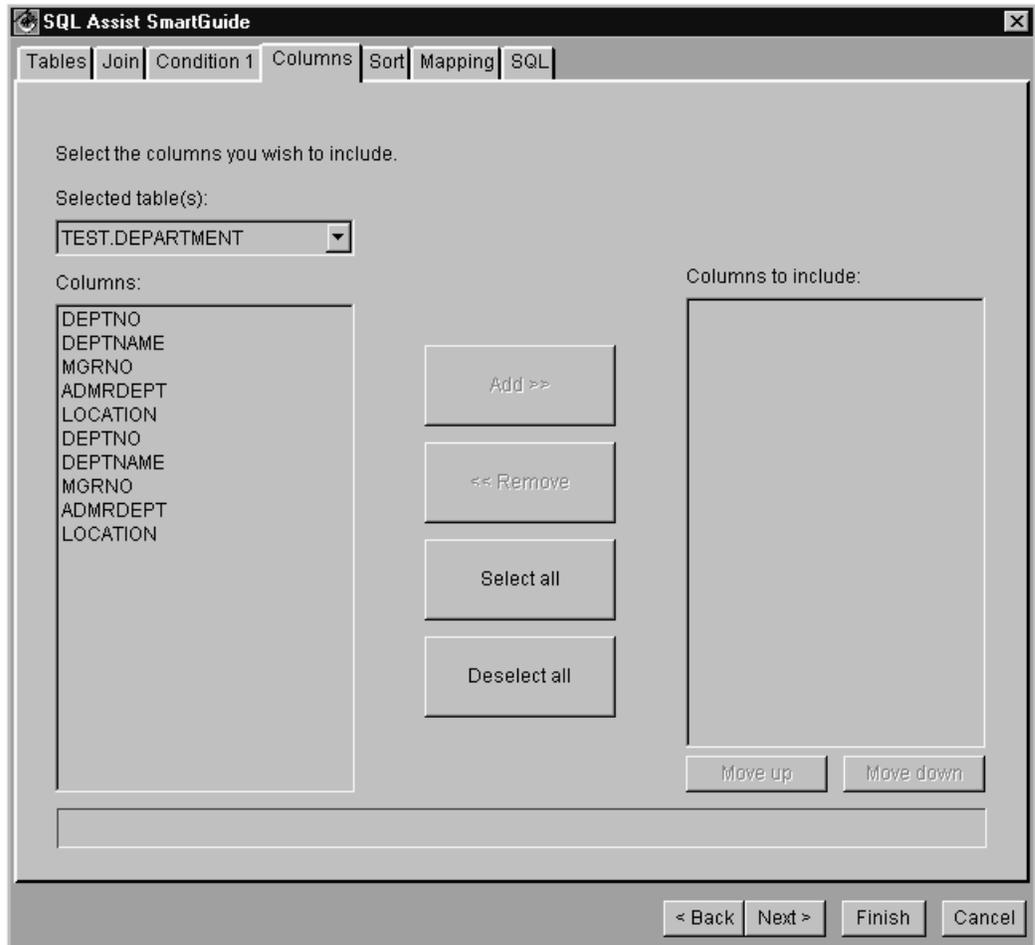
1. Select a table from the **Selected table(s)** drop-down list. The list includes only the tables that are selected in the Tables page.

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 moved from the **Columns** list to the **Columns to include** list.

   To remove one or more columns from the **Columns to include** list, select the columns and select **Remove**. The selected columns are moved from the **Columns to include** list to the **Columns** list.

   To change the order of columns in the **Columns to include** list, select the columns whose position you would like to change and then select **Move up** or **Move down**.

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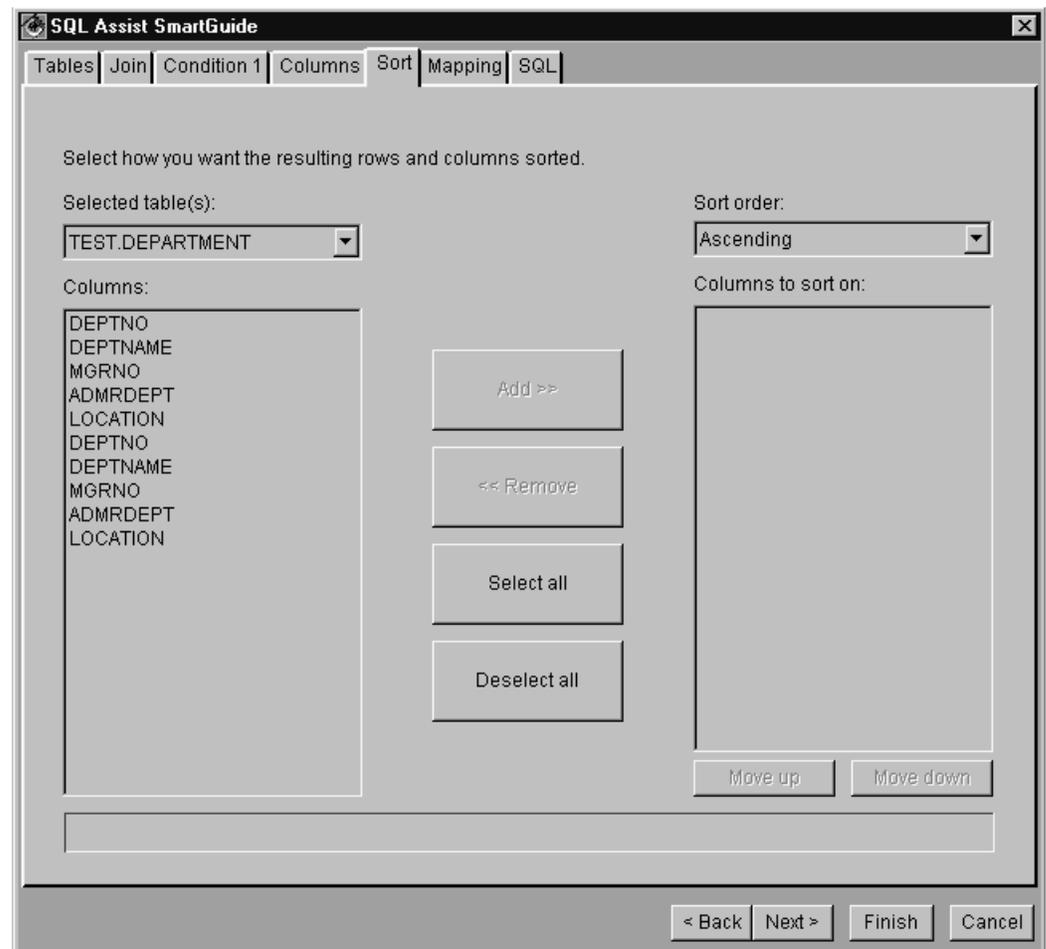
1. Select a table from the **Selected table(s)** drop-down list. The list includes only the tables that are selected in the Tables page.
2. Select one or more columns from the **Columns** list. The list includes the columns for the selected table. Use **Select all** to select all the columns in the list.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

RELATED TASKS
Specifying the tables for an SQL statement
Joining tables
Specifying search conditions
Specifying result columns
Remapping data to a different SQL data type
Displaying the SQL statement

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[]

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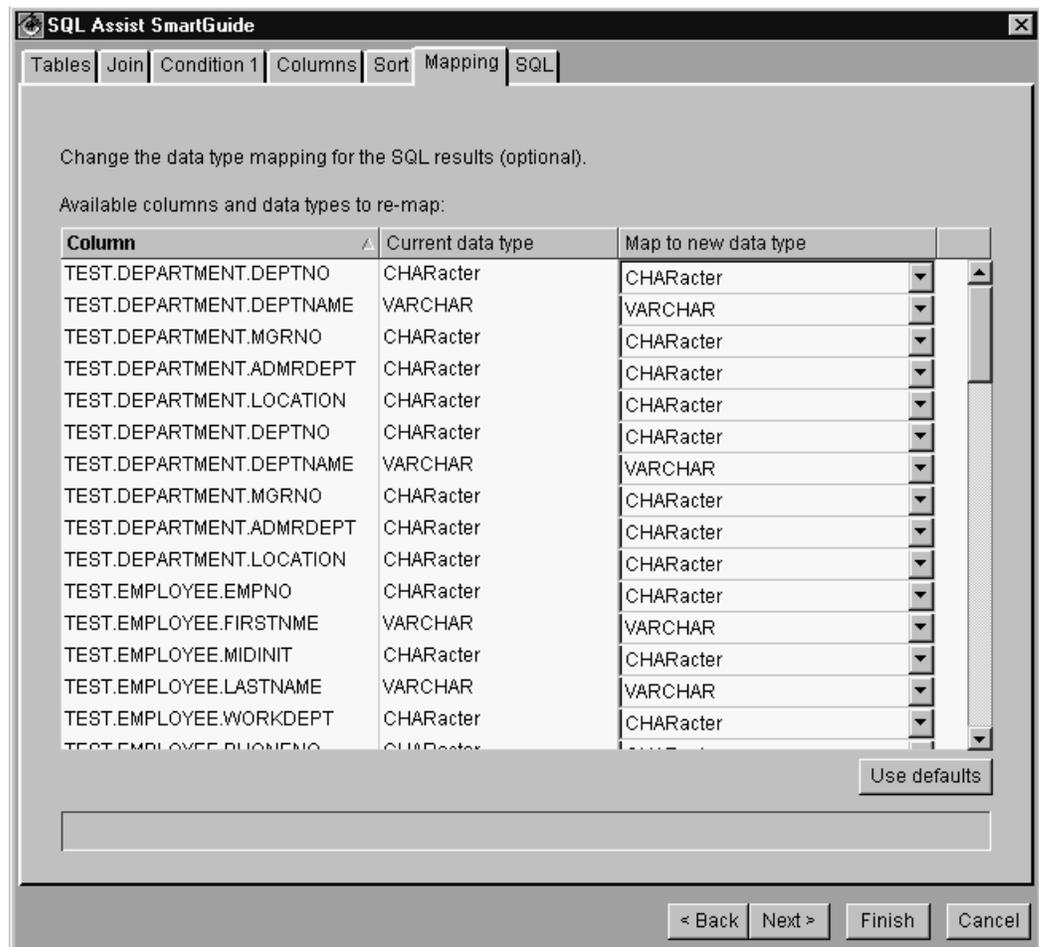
1. Select a column from the **Column** list. **Current data type** displays the SQL data type for the column.
2. Select an SQL data type from the drop-down list in **Map to new data type**. The data retrieved from the column will be mapped to the selected SQL data type.

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**.

RELATED TASKS
Specifying the tables for an SQL statement
Joining tables
Specifying search conditions
Specifying result columns
Sorting the result set
Displaying the SQL statement

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

```
SQL Assist SmartGuide                          [X]

Tables | Join | Condition 1 | Columns | Sort | Mapping | SQL |


   View the SQL statement.

   SELECT                                     [▲]
       MOJO.DEPARTMENT.DEPTNO,
       MOJO.DEPARTMENT.DEPTNAME
   FROM
     MOJO.DEPARTMENT




                                              [▼]
   [◄]                                   [►]

   ☑ Schema qualified names

   [ Copy to clipboard ]  [ Save SQL... ]  [ Run SQL... ]

   [                                          ]


              < Back | Next > | Finish | Cancel
```

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.

Schema qualified names

The schema qualified names checkbox allows you to include the schema name on tables and columns by selecting it. At runtime, you can use the same database and user ID you used to specify the statement, or another database and/or user ID. If the **Schema qualified names** checkbox is selected, the tables must be defined on the database where the statement executes, with the same schemas you specified. If

the checkbox is not selected, the tables must be defined on the database where the statement executes, with a schema that matches the user ID you use to connect.

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 if one is returned from executing the SQL statement. .

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**.

RELATED TASKS
Specifying the tables for an SQL statement
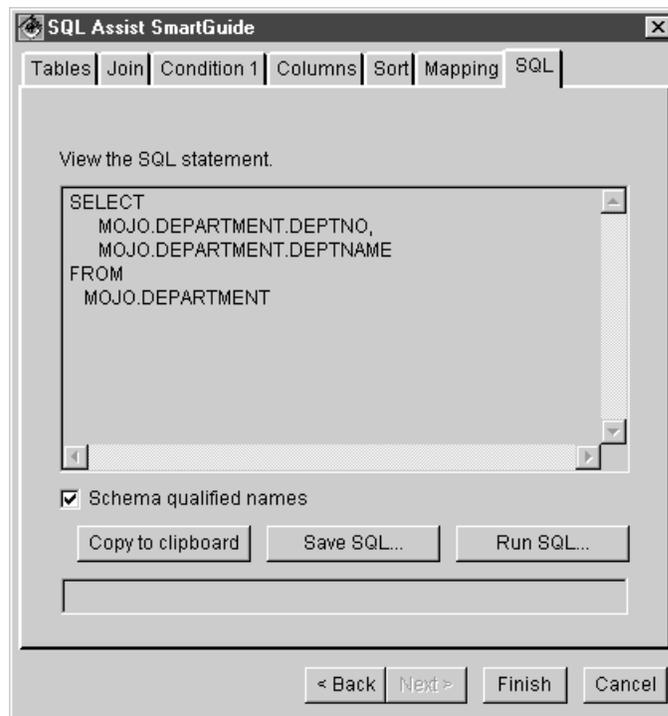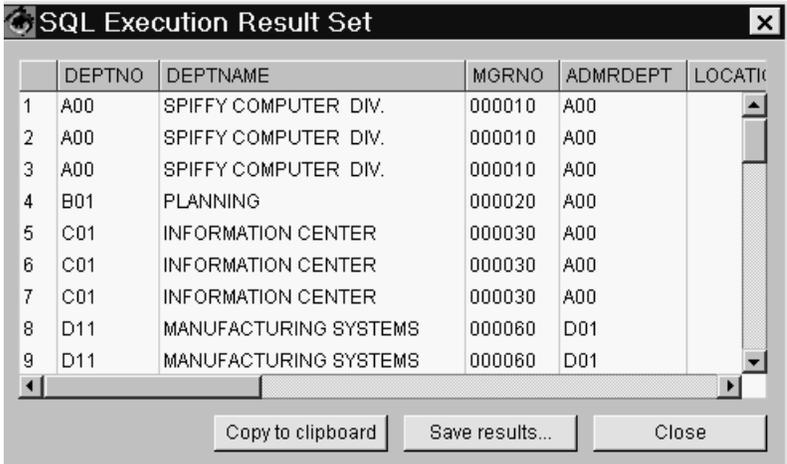Joining tables
Specifying search conditions

Specifying result columns
Sorting the result set
Remapping data to a different SQL data type

Composing an SQL INSERT, UPDATE, or DELETE visually

Use the SQLAssist SmartGuide to visually compose an SQL INSERT, UPDATE, or DELETE for a Modify bean.
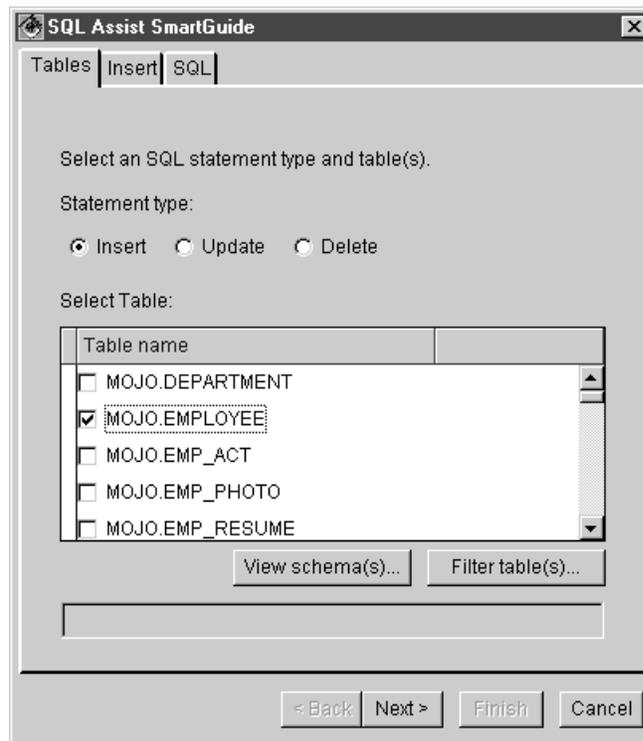
Using the SQLAssist SmartGuide you can:

- Specify the table to be accessed in the SQL insert, update, or delete. (Required.)
- Specify the table columns and their new values for the SQL update or insert. (Optional.)
- Specify search conditions for the SQL update or delete. (Optional.)
- Display the resulting SQL insert, update, or delete. (Optional.)

You must first chose the statement type. Chose the **Insert** button if you want insert data into the database. Chose the **Update** button if you want to update rows in the database. Chose the **Delete** button if you want to delete rows in the database.

When you display the SQL INSERT, UPDATE, or DELETE, 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
Specifying columns and their values for an SQL UPDATE or INSERT
Specifying search conditions
Displaying the SQL statement

Specifying columns and their values for an SQL UPDATE or INSERT

Using the SQL Assist SmartGuide Insert/Update page, you can specify columns and their values.

To specify columns and values for an insert:

1. From the SQL Assist SmartGuide, click the **Insert** button and choose the table where you want to insert.
2. Click **Next.** The Insert window displays.
3. Specify values for the columns that you want to insert in the **Value** list and click **Next**.   The SQL statement displays.
4. Click **Finish** to generate the INSERT SQL statement.
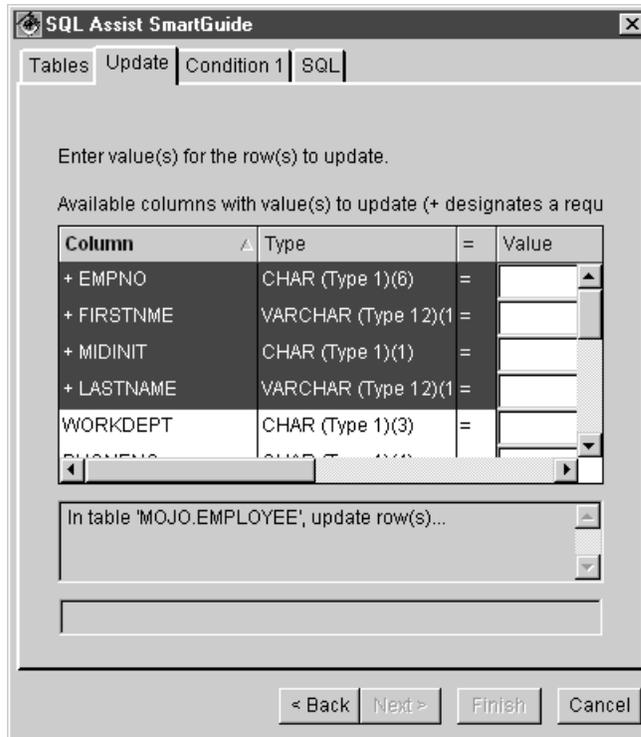
To specify columns and values for an update:

1. From the SQL Assist SmartGuide, click the **Update** button and choose the table you want to update.
2. Click **Next.** The Update window displays.
3. Specify values for the columns that you want to update in the **Value** list and click **Next**. You can specify conditions for the SQL statement.
4. Click **Finish** to generate the UPDATE SQL statement.

You can also specify parameters in the **Value** list. If a parameter is specified, its value is used in the INSERT or UPDATE statement. 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.

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

RELATED TASKS
Composing an SQL INSERT, UPDATE or DELETE
Specifying the tables for an SQL statement
Specifying columns and their values for an SQL UPDATE or INSERT
Specifying search conditions
Displaying the SQL statement

Composing an SQL procedure call visually

Use the SQL Assist SmartGuide for Stored Procedures to visually compose an SQL stored procedure call for a ProcedureCall bean.
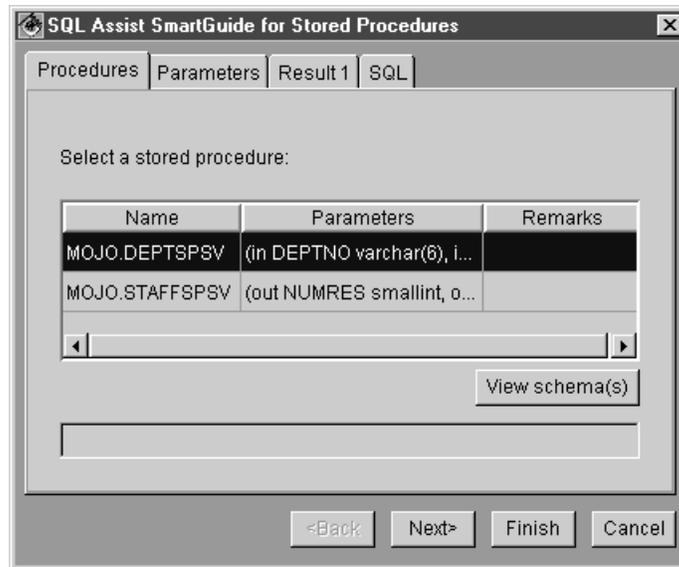
Using the SQLAssist SmartGuide you can:
- Specify the stored procedure to execute. (Required.)
- Specify hard-coded values for input parameters to the stored procedure. (Optional.)
- Describe any result sets returned by the stored procedure. (Optional.)
- Display the resulting SQL call statement. (Optional.)

RELATED TASKS
Specifying the stored procedure for an SQL CALL statement
Working with stored procedure parameters
Defining result sets returned by a stored procedure
Displaying the SQL statement

Specifying the stored procedure for an SQL CALL statement
Use the Procedures page of the SQL Assist SmartGuide for Stored Procedures to specify the stored procedure that is executed in an SQL CALL statement.
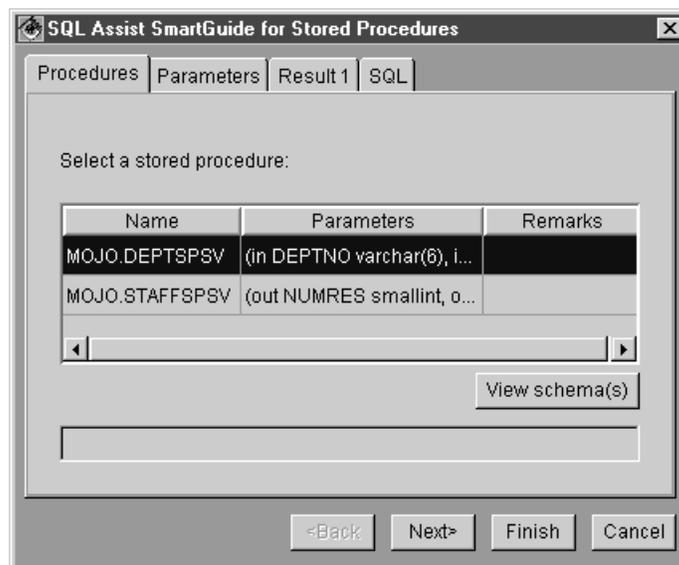
Each stored procedure you can access in the database identified by the currently selected connection alias is listed in a row of the table on this page.

- The **Name** field shows the schema-qualified name of the stored procedure.
- The **Parameters** field shows the procedure's parameters, including their names, SQL data types, and modes (in, out, inout).
- The **Remarks** field shows any remarks registered for the stored procedure.

By default, the stored procedures listed are those whose schema is the user ID specified for the database connection. If no procedures in the database have that schema, all the procedures in the database are listed.

You can add to or remove from the stored procedure names displayed in the list, by selecting **View schema(s)**.

Selecting **View schema(s)** opens the Schema(s) to View window.
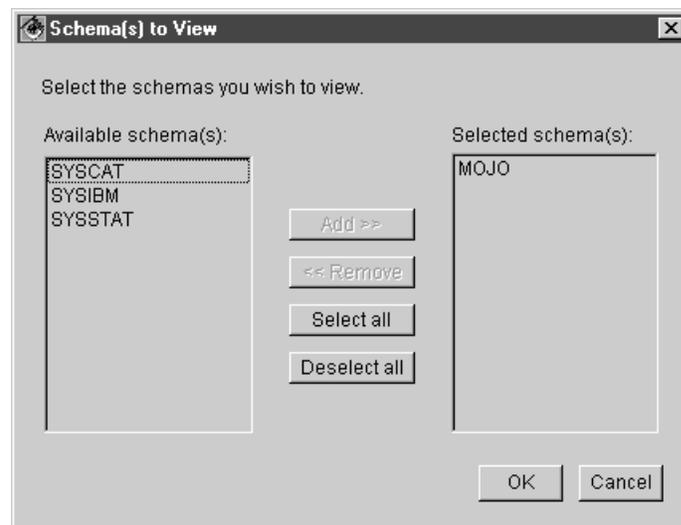
- You can select schemas from Available schema(s) and add them to Selected schema(s) by clicking the **Add** button.
- You can remove schemas from Selected schema(s) using the **Remove** button.
- You can select or deselect all the schemas in both Available schema(s) and Selected schema(s) using the **Select all** or **Deselect all** buttons.

When you finish selecting schemas to view, select **OK**. This closes the Schema(s) to View window.

Select a stored procedure for the SQL CALL statement by clicking on the row for that stored procedure. You can select only one stored procedure.

When you finish, select **Next**. This displays the Parameters page of the SQL Assist SmartGuide for Stored Procedures. Use this page to work with the parameters for the selected stored procedure. You can also display any page in the SQL Assist SmartGuide for Stored Procedures by selecting its tab.

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

Working with stored procedure parameters

Use the Parameters page of the SQLAssist SmartGuide for Stored Procedures to examine the parameters registered for the stored procedure and to optionally hard-code an input value for parameters whose mode is in or inout.
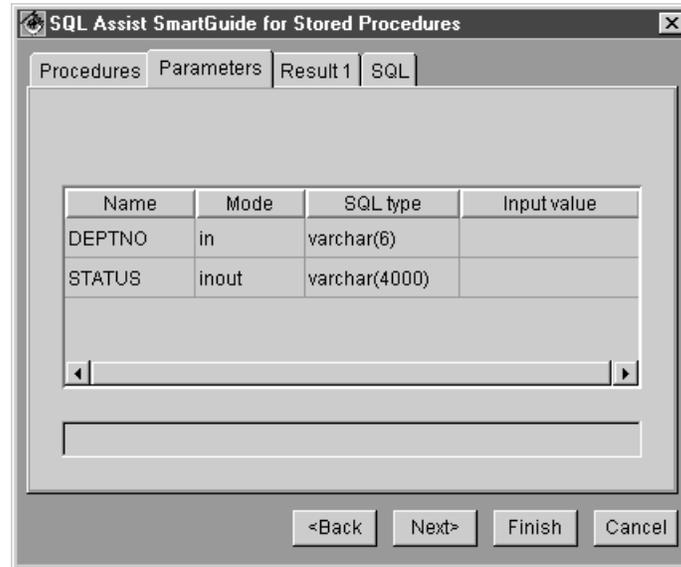
Each parameter registered for the stored procedure selected on the Procedures page is described in a row of the table on this page.

- The **Name** field shows the name of the parameter.
- The **Mode** field shows whether the parameter is used for input, output, or both input and output.
- The **SQL type** field shows the SQL data type of the parameter. The field also shows the user-specifiable length and scale for SQL data types where this is meaningful.
- The **Input value** field shows any input values you have hard-coded for parameters whose mode is in or inout. If the mode of a parameter is out, the value None appears in this field. If the mode of a parameter is in or inout and you have not hard-coded a value, nothing appears in this field, and you will specify the input value at run-time by setting a parameter on the CALL statement.

VisualAge for Java generates a parameter in the CALL statement for each parameter that does not have a hard-coded input value. The name of the parameter is the same as the name that appears in the **Name** field.

To hard-code an input value for a parameter, enter the desired value in the **Input value** field for that parameter. For a parameter whose mode is output, that field is not editable. For a parameter whose mode is inout, the field is editable, but you should generally not hard-code an input value. If you do, no parameter is generated in your CALL statement, and you cannot obtain the output value after the stored procedure executes.

When you finish working with parameters, select **Next**. This displays the first Result page of the SQL Assist SmartGuide for Stored Procedures. Use this page to describe the first result set if the stored procedure returns at least one result set.

Select **Back** to display the Procedures page of the SQL Assist SmartGuide for Stored Procedures. If appropriate, you can then change the stored procedure that is executed.

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

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

Defining result sets returned by a stored procedure

Use the Result page of the SQL Assist SmartGuide for Stored Procedures to definee a result set returned by the stored procedure selected on the Procedures page. You can define multiple result sets. You define each result set on a separately numbered Result page.
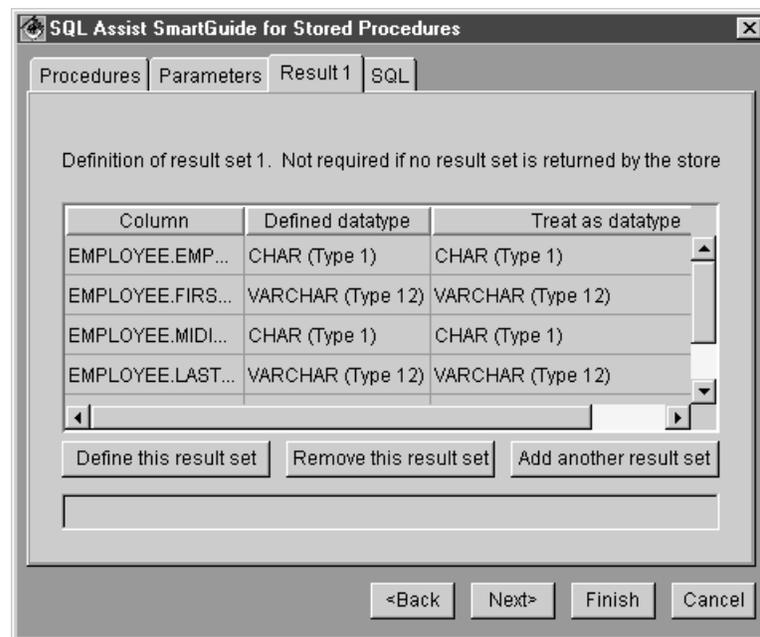
Defining a result set

For the current numbered Result page, if you have already defined this result set, each of its columns is described in a row of the table on this page; if you have not yet defined this result set, the table on this page is empty.

For each column:
- The **Column** field shows the name of the column, qualified by the table name.
- The **Defined datatype** field shows the SQL data type of the column in the database.
- The **Treat as datatype** field shows the SQL data type to which you have mapped the column. (By default, this is the same as **Defined datatype**.)

To define this result set for the first time, or to modify your previous definition of this result set, select **Define this result set**. This opens the SQL Assist SmartGuide to its Tables page. See "Defining a result set visually" for instructions on using the

SQL Assist SmartGuide to define a result set. After you have finished defining the result set, your new definition will be shown in the table on this page.

If the stored procedure does not return any result sets, leave the table on the first numbered Result page empty, and do not define any additional result sets.

Defining additional result sets

From the first Result page (the tab for the page is labeled Result 1) select **Add another result set**. This displays a second Result page labeled Result 2. Define the second result set as described in "Defining a result set (page 36)."

Define a third result set by selecting **Add another result set** in the Result 2 page. Repeat the process until you define all the result sets returned by the stored procedure.

You need not complete the definition of one result set before adding another one. You can return to the page for any result set and complete its definition at a later time.

If the stored procedure returns multiple result sets, but their definitions are identical, you only need to define one result set.

Removing a result set description

Select the Result page for the result set. Then select **Remove this result set**. The Result page will be removed from the notebook and any Result pages that follow it will be renumbered to reflect its removal.

When you finish defining a result set

When you finish defining one result set, select **Next** to display either the next result set (if you have added another one) or the SQL page of the SQL Assist SmartGuide for Stored Procedures. Use the SQL page to view the SQL CALL statement.

Select **Back** to display either the previous result set (if this is not Result 1) or the Parameters page of the SQL Assist SmartGuide for Stored Procedures.

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

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

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

RELATED TASKS
Defining a result set visually
Specifying the stored procedure for an SQL CALL statement
Working with stored procedure parameters
Displaying the SQL statement for a ProcedureCall bean

Displaying the SQL statement for a ProcedureCall bean

Use the SQL page of the SQL Assist SmartGuide for Stored Procedures to display the SQL statement for a ProcedureCall bean.
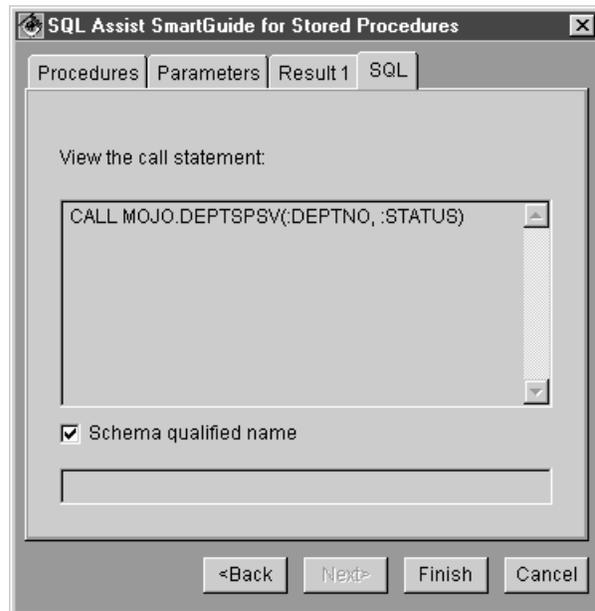
Schema qualified name

The schema qualified name checkbox allows you to qualify the stored procedure name with its schema name by selecting it. If you have described result sets that the stored procedure returns, selecting this checkbox also causes table names in the generated meta-data to be qualified with their schema names. At runtime, you can use the same database and user ID you used to specify the CALL statement, or another database and/or user ID. If the **Schema qualified name** checkbox is selected, the stored procedure and tables must be defined on the database where the statement executes, with the same schemas you specified. If the checkbox is not selected, the stored procedure and tables must be defined on the database where the statement executes, with a schema that matches the user ID you use to connect.

You can edit any part of the SQL specification by selecting the appropriate page in the SQL Assist SmartGuide for Stored Procedures. Selecting **Back** displays the last Result page. If appropriate, you can use this page to re-specify the result set.

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

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

Composing an SQL statement manually

Use the SQL Editor to manually define an SQL statement for a Select, Modify, or ProcedureCall 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.) If you manually define an SQL statement, you may need to add meta-data information to the SQL specification generated in your database access class.
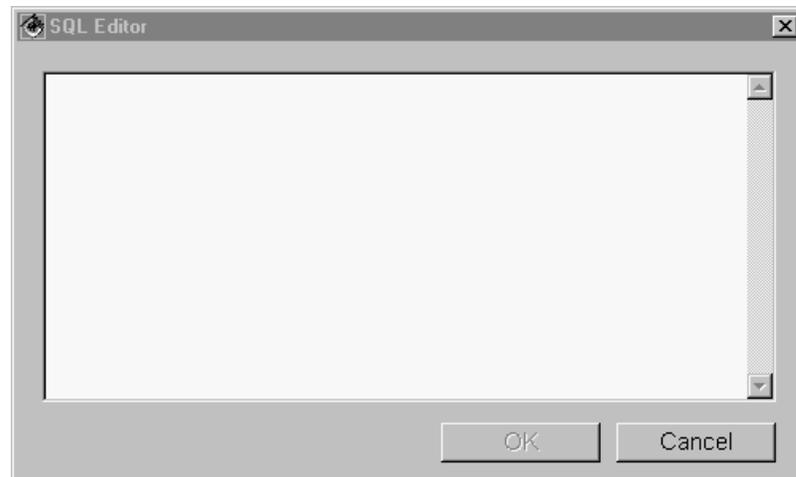
As you compose your SQL statement, you can:

- 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**.

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. To open the property sheet of a Select bean on the Visual Composition Editor surface:

1. Right click the **Select** bean. The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

newclass2 - Properties

Select1

beanName	Select1
distinctTypesEr	False
fillCacheOnExe	True
forceSearchedL	False
lockRows	False
maximumPacke	0
maximumRows	0
packetSize	1
query	
readOnly	False
timeout	0
validateLOBs	False

beanName

Reset...

☑ Show expert features
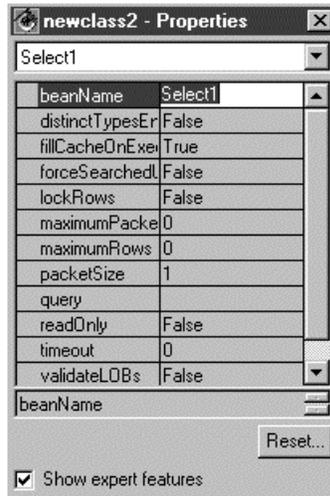
You can control:

- The name of the Select bean. Specify the name as the **beanName** property value.
- Whether to enable inserts, updates, and deletes for result sets which contain user-defined (distinct) types. Specify this in the **distinctTypesEnabled** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether as many rows as possible should be fetched into the cache as soon as you execute your query (given your settings for the above properties), or rows should be fetched only as you ask for them. Specify this in the **fillCacheOnExecute** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether to force generation of searched rather than positioned SQL UPDATE and DELETE statements for the result set returned by the statement. Specify this in the **forceSearchedUpdate.** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether to acquire and hold a database lock for a row while it is the current row. Specify this in the **lockRows** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of packets allowed in the cache at one time. (Older packets may be displaced as newer packets are fetched.) Specify this in the **maximumPacketsInCache** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of rows you can fetch into the cache over time. Specify this in the **maximumRows** property value. Check the **Show Expert Features** checkbox to display this property.
- The number of rows in a packet. Specify this in the **packetSize** property value. Check the **Show Expert Features** checkbox to display this property.
- The database connection characteristics and SQL statement. Specify these as a composite value for the **query** property. See Specifying a Connection and SQL Statement.
- Whether the result set is updatable. Specify this in the **readOnly** property value.
- The maximum number of seconds allowed for the statement to execute. Specify this in the **timeout** property value. The default is 0, which means no maximum. Check the **Show Expert Features** checkbox to display this property.
- Whether to validate and, if necessary, re-fetch Clob and Blob objects before returning them to you as column or parameter values.

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.

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 which Select or ProcedureCall bean the DBNavigator bean will navigate. When selected, the Execute button in the DBNavigator bean invokes the execute method of the Select bean, which executes the SQL statement.

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. Because the text property is not bound, you must also specify in the connection properties an event to trigger the propagation of the text value to the parameter. Once you do this, code is generated to invoke the setParameterFromString method whenever the event is fired.

When you execute an SQL statement using a Select bean, it returns a result set. Unlike the native Java interface to relational data (JDBC), the Select bean maintains rows of the result set in a memory cache where you can move back and forth among the rows. The number of rows initially fetched when you execute is controlled by the following properties of the Select bean:

- fillCacheOnExecute
- maximumRows
- packetSize, (Rows are fetched in groups called packets)
- maximumPacketsInCache

If fillCacheOnExecute is set to false, one packet of rows is initially fetched. If fillCacheOnExecute is set to true, the number of rows fetched is the least of: all of the rows, the limit imposed by the maximumRows property, the limit imposed by packetSize times maximumPacketsInCache. (If you use the default values for all of these properties, all of the rows are fetched.)

If there are any rows in the result set, regardless of how many rows were initially fetched, you are positioned on the first one.

Editing Modify bean properties

You specify and control the operation of the Modify bean by opening the property
sheet for the Modify bean and setting the value of the displayed properties. To
open the property sheet of a Modify bean on the Visual Composition Editor
surface:

1. Right click the **Modify** bean.   The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

You can control:

- The database connection characteristics and SQL statement. Specify these as a
  composite value for the **action** property. See "Specifying a connection and SQL
  statement."
- The name of the Modify bean. Specify the name as the **beanName** property
  value.
- The maximum number of seconds allowed for the statement to execute. The
  default value is 0, which means no maximum. Specify this in the **timeout.**
  property value. Check the **Show Expert Features** checkbox to display this
  property.
- Whether to validate and, if necessary, re-fetch Clob and Blob objects before
  returning them to you as column or parameter values.

Executing a Modify bean

To access relational data using a Modify bean, you connect an interface component to the Modify bean. For example, you can make an event-to-method connection between the actionPerformed event for a button and the execute method of the Modify bean. When the button is selected, the SQL statement associated with the Modify 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. Because the text property is not bound, you must also specify in the connection properties an event to trigger the propagation of the text value to the parameter. Once you do this, code is generated to invoke the setParameterFromString method whenever the event is fired.

The purpose of the Modify bean is to execute SQL INSERT, UPDATE, and DELETE statements. After you execute, the value of the numAffectedRows property tells you how many rows were inserted, updated, or deleted. It is possible to execute other kinds of SQL statements using a Modify bean, but if the statement produces a result set, the Modify bean does not give you access to it.

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

RELATED TASKS
"Editing Select bean properties" on page 39
Connecting beans

RELATED REFERENCES
"Chapter 3. Data Access Beans" on page 75

Editing ProcedureCall bean properties

You specify and control the operation of the ProcedureCall bean by opening the property sheet for the ProcedureCall bean and setting the value of the displayed properties. To open the property sheet of a ProcedureCall bean on the Visual Composition Editor surface:

1. Right click the **ProcedureCall** bean. The bean menu displays.
2. Click **Properties**. The Properties sheet displays.
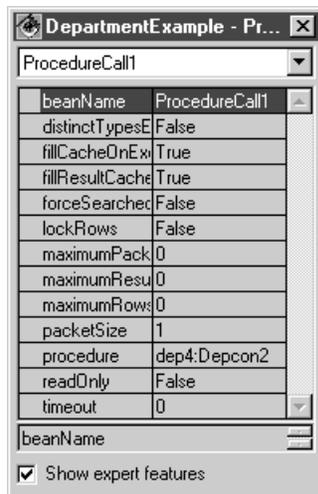
You can control:

- The name of the ProcedureCall bean. Specify the name as the **beanName** property value.
- Whether to enable inserts, updates, and deletes for result sets which contain which contain user-defined (distinct) types. Specify this in the **distinctTypesEnabled** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether as many rows as possible should be fetched into the cache as soon as you execute your stored procedure or rows should be fetched only as you ask for them. Specify this in the **fillCacheOnExecute** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether as many result sets as possible should be fetched into the cache as soon as you execute your stored procedure (given your settings for the above properties), or result sets should be fetched only as you ask for them. Specify this in the **fillResultCacheOnExecute** property value. Check the **Show Expert Features** checkbox to display this property.
  If you set this property to true, it is advisable to set the **fillCacheOnExecute** property to true as well. Otherwise, there may be rows in the result sets, other than the last result set, that can never be fetched.
- Whether to force generation of searched rather than positioned SQL UPDATE and DELETE statements for result sets returned by the stored procedure. Specify this in the **forceSearchedUpdate.** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether to acquire and hold a database lock for a row while it is the current row. Specify this in the **lockRows** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of packets allowed in the cache at one time for each result set. (Older packets may be displaced as newer packets are fetched.) Specify this in the **maximumPacketsInCache** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of result sets allowed in the cache at one time. (Older result sets may be displaced as newer ones are fetched.) Specify this in the **maximumResultsInCache** property value. Check the **Show Expert Features** checkbox to display this property.

- The maximum number of rows you can fetch into the cache over time. Specify this in the **maximumRows** property value. Check the **Show Expert Features** checkbox to display this property.
- The number of rows in a packet. Specify this in the **packetSize** property value. Check the **Show Expert Features** checkbox to display this property.
- The database connection characteristics and SQL statement. Specify these as a composite value for the **procedure** property. See Specifying a Connection and SQL Statement.
- Whether the result set is updatable. Specify this in the **readOnly** property value.
- The maximum number of seconds allowed for the statement to execute. Specify this in the **timeout** property value. The default is 0, which means no maximum. Check the **Show Expert Features** checkbox to display this property.

RELATED TASKS

Specifying a connection and SQL statement.

RELATED REFERENCES

ProcedureCall

Executing a ProcedureCall bean

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

Alternatively, you can connect the DBNavigator bean to the ProcedureCall 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 ProcedureCall bean and the model property of the DBNavigator bean. The this property refers to the whole object of the ProcedureCall bean. The model property specifies which Select or ProcedureCall bean the DBNavigator bean will navigate. When selected, the Execute button in the DBNavigator bean invokes the execute method of the ProcedureCall bean, which executes the SQL statement.

If you have defined input parameters in your SQL statement, you must set the parameters before you invoke the execute method. Likewise, if you have defined output parameters, you will want to get their values after 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 (input or output) 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. Because the generated property is bound, when you make the connection, code is generated to invoke the getParameterFromString method whenever the parameter value changes. This is sufficent for an output parameter. For an input parameter, because the text property is not bound, you must also specify in the connection properties an event to trigger the propagation of the text value to the parameter. Once you do this, code is generated to invoke the setParameterFromString method whenever the event is fired.

When you execute an SQL statement using a ProcedureCall bean, it may return no result sets, one result set, or many result sets. Unlike the native Java interface to relational data (JDBC), the ProcedureCall bean maintains rows of the result sets in a memory cache where you can move back and forth both among different result sets and among the rows of each result set.. The number of result sets and the number of rows in each that are initially fetched when you execute are controlled by the following properties of the ProcedureCall bean:

- fillCacheOnExecute
- fillResultCacheOnExecute
- maximumResultsInCache
- maximumRows
- maximumPacketsInCache
- packetSize (Rows are fetched in groups called packets.)

If fillResultCacheOnExecute is set to false, one result set is initially fetched. If it is set to true, the number of result sets fetched is the lesser of all result sets and the limit imposed by maximum ResultsInCache.

If fillCacheOnExecute is set to false, one packet of rows is initially fetched. If it is set to true, the number of rows fetched for each result set is the least of: all of the rows, the limit imposed by the maximumRows property, the limit imposed by packetSize times maximumPacketsInCache.

(If you use the default values for all of these properties, all of the rows in all of the result sets are fetched.)

If there are any result sets, regardless of how many result sets were initially fetched, you are positioned on the first one; and if the result set has any rows, regardless of how many of its rows were fetched, you are positioned on the first one.

For some database products, it may be necessary for you to fetch all of the result sets returned by a stored procedure before getting the values of any output parameters. The simplest way to accomplish this is to use the default values for the fillResultCacheOnExecute and maximumResultsInCache properties. This way all of your result sets will be immediately fetched into the cache.

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

RELATED TASKS
"Editing Select bean properties" on page 39
"Adding the DBNavigator bean to the Visual Composition Editor surface" on page 69
Connecting Beans

RELATED REFERENCES
"Chapter 3. Data Access Beans" on page 75

Specifying a connection and SQL statement

The query property of a Select bean, the action property of a Modify bean and the procedure property of a ProcedureCall bean are three examples of what we generically refer to as data properties. The two parts of a data property are:

Connection alias

Specifies database connection characteristics for the bean

SQL specification

Specifies an SQL statement for the bean

To edit a data property:

1. Open the property sheet for a Select, Modify or ProcedureCall bean by double-clicking on the **bean**.

2. Select the **query**, **action** or **procedure** property value and click on the box in the value field. The data property editor displays.
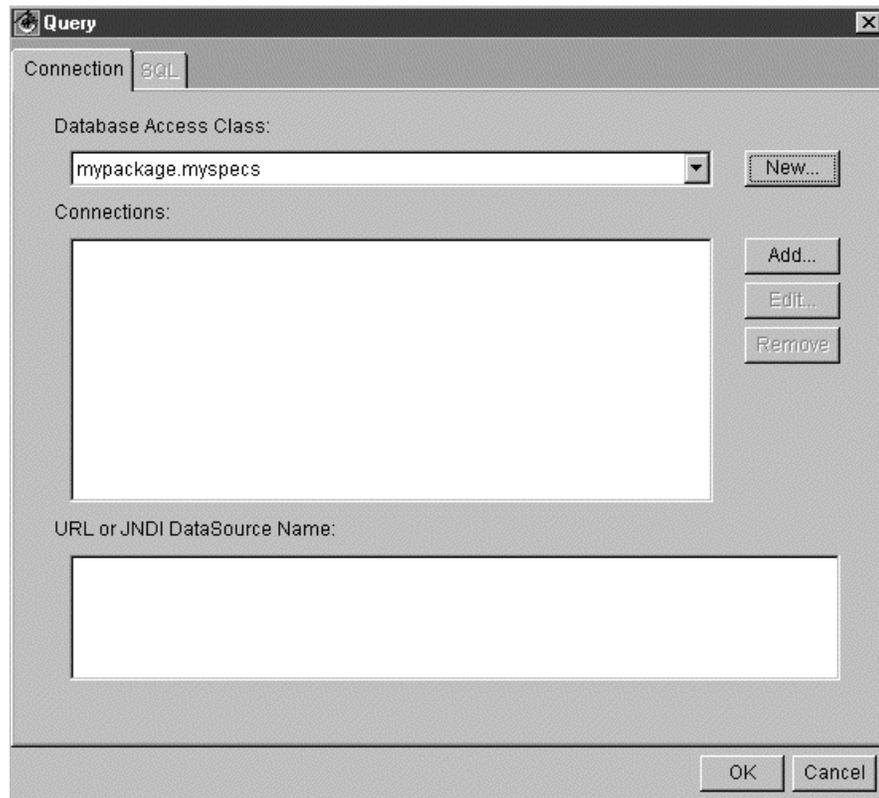
3. Create a connection alias or select an existing connection alias.

4. Create an SQL specification or select an existing SQL specification.

RELATED TASKS

"Specifying a connection alias"

"Making an SQL specification" on page 12

Specifying a connection alias

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

Use the Connection page of the data 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, Modify or ProcedureCall beans can use the same connection alias, and if so, they share the database connection associated with that connection alias at runtime. If one bean commits updates to a database, it commits all uncommitted updates made by any bean sharing the database connection.
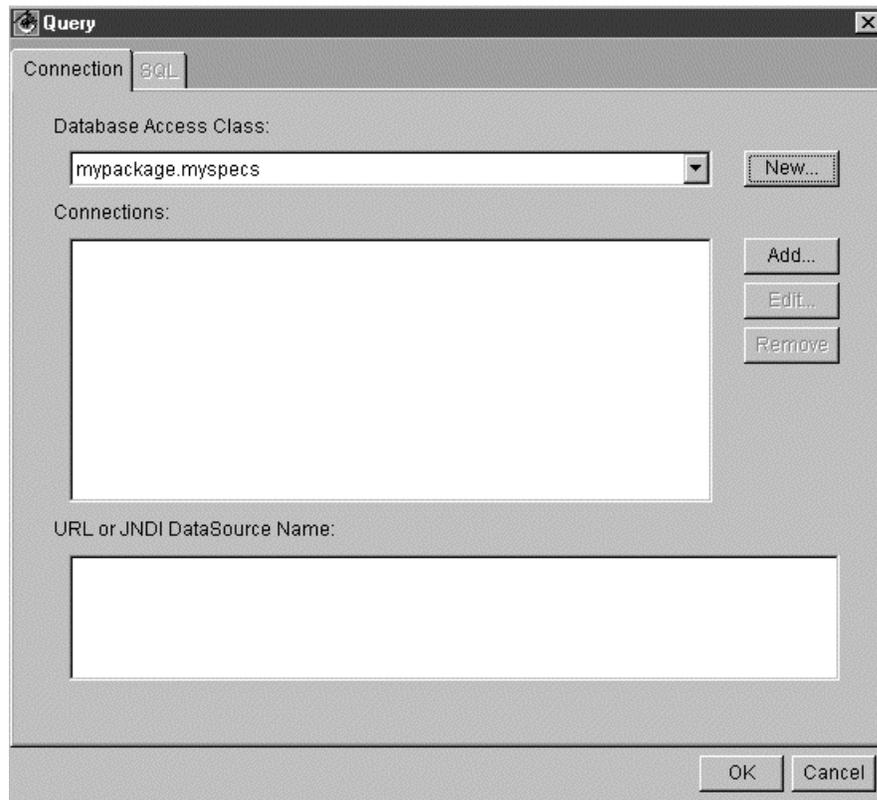
Select a database access class from the drop-down list in the **Database Access Class** field. The list shows the database access classes that exist in the workspace. The format of each item in the list is *packagename.classname*, where *packagename* is the name of the package that contains the database 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**.
See Defining a Database Access Class.

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 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**.

Defining or editing a connection alias

A connection alias specifies the characteristics of a database connection for a Select, Modify, or ProcedureCall bean. Select **Add** in the Connection page of the data property editor to define a new connection alias. Select a connection alias from the **Connections** list in the Connection page of the data property editor and select **Edit** to edit the selected connection alias. Selecting **Add** or **Edit** opens the Connection Alias Definition window. You can specify most of the information about a connection on the Basic page of the Connection Alias Definition window.

- Specify a name for the connection alias in the **Connection Name** field. The name must be a valid Java method name. Although the list on the Connection page of the data property editor contains only connection alias definitions, the name you provide here must be unique within all of the definitions in the database access class, including other connection alias definitions and SQL specification definitions.
- Click the radio button next to the source you want to obtain a connection from. The Connection Alias Definition window will change depending on the connection source you select. To obtain a connection from the DriverManager object, your connection alias must contain all of the information needed to connect to a database. To obtain a connection from a DataSource object, the DataSource object itself must contain the information needed to connect, and your connection alias only has to contain enough information to find the DataSource object, which should have been previously registered in a JNDI naming service.

 Note: To use a DataSource object, you must be using a JDBC 2.0 driver.

If you select the default connection source, DriverManager:
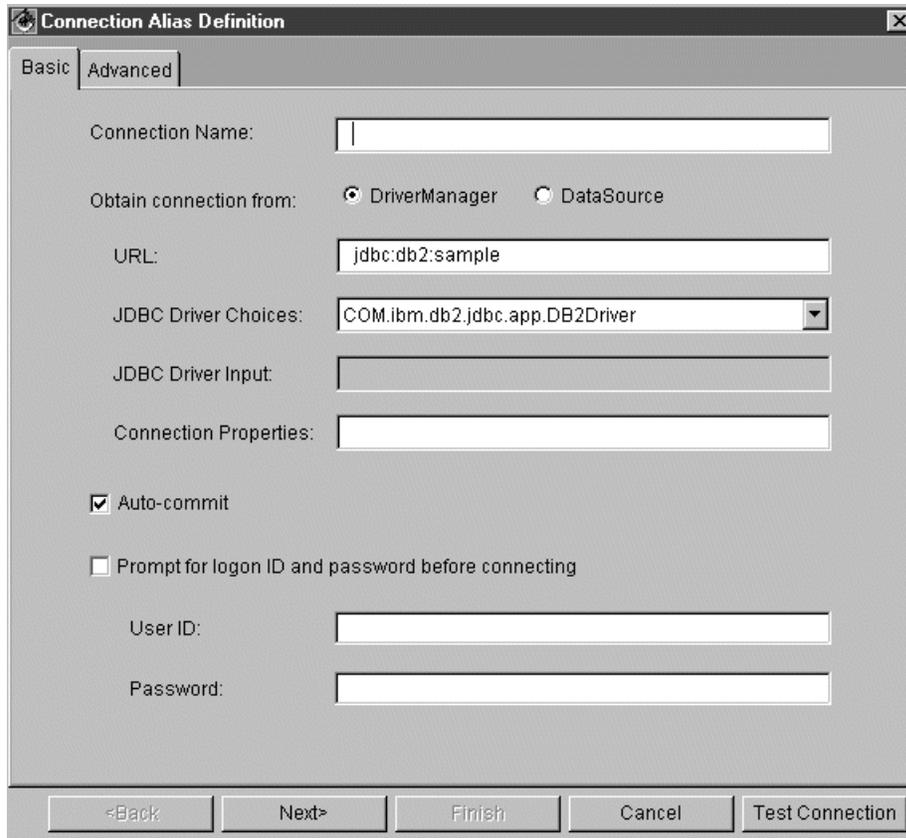
- Specify the URL for the database connection in the **URL** field. The URL specification must be in the format jdbc:*subprotocol*:*subname*, where *subprotocol* and *subname* identify the DataSource for the connection. The value of *subprotocol* depends on the JDBC driver used. For example, for the 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 *subname* value.

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

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

  Here, the *subname* 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
  ```

- Select the **Auto-commit** check box if you want database updates to be automatically committed for each SQL statement. If you do not select the check box, database updates are not automatically committed. This check box is selected by default.

- Select the **Prompt for logon ID password before connecting** check box if you want the user to be prompted for the user ID and password to be used in the database connection request. Do not select the check box 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** check box is not selected.

- 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** check box is not selected.

If you select the alternate connection source, DataSource:

- Select from the **Initial Context Factory** list the initial context factory for the naming service you want to use. Four initial context factory values are provided for you.

COM.ibm.db2.jndi.DB2InitialContextFactory
Select this value to use the DB2 naming service.

com.ibm.ejs.ns.jndi.CNInitialContextFactory
Select this value to use the WebSphere™ naming service.

Use value of java.naming.factory.initial property
Select this value if you have set the java.naming.factory.initial property in your java environment to identify a default initial context factory, and you wish to use that default.

Note: This property can not be set inside the VisualAge for Java environment. However, you may still want to choose this option if the property is set in your deployment environment.

Use other Initial Context Factory
Select this value to enter the name of a context factory not listed here. After selecting this value, type the name of the context factory you want to use in the **Initial Context Factory** field.

- Specify in the **Provider URL** field the URL of the machine where the naming service should look for your DataSource if your DataSource is located on another machine.
- Select from the **DataSource Name** list, the name of the DataSource you want to use. The **DataSource Name** list includes all DataSource objects found using the initial context factory and provider URL you have specified. If you know that a DataSource not listed here will be found in your deployment environment, you can type that DataSource name into this field.
- Click **Maintain DB2 DataSources** if you want to add, edit, or remove DB2 DataSource objects registered with a naming service. The Maintain DB2 DataSources window opens.

  Since only DB2 DataSource objects can be maintained with this window, the initial context factory for the naming service defaults to **COM.ibm.db2.jndi.DB2InitialContextFactory**. However, you can maintain DB2 DataSources registered with another naming service by typing in a different initial context factory.
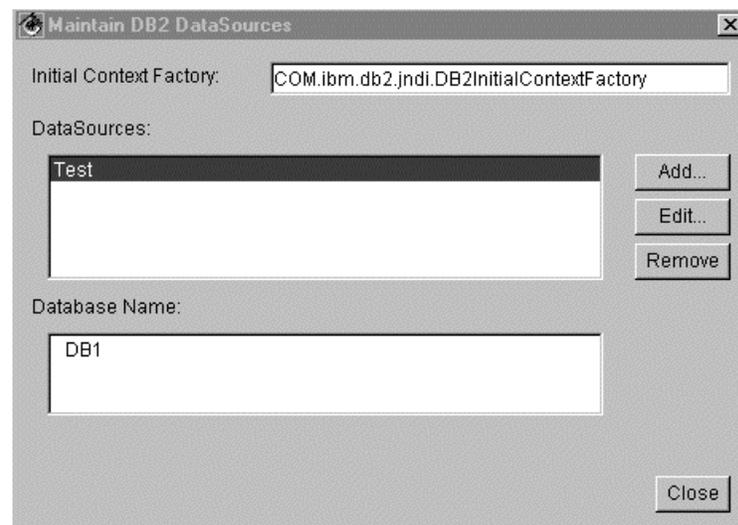
- Select from the list in the **DataSources** text box the DataSource you want to maintain. To browse or edit more information on a listed DataSource, select the

DataSource name from the list, and click **Edit** to display the Edit DB2
DataSource window. The fields in this window are the same as those in the
Add DB2 DataSource window.. If you want to add a DataSource not already
listed, click **Add** to open the Add DB2 DataSource window.
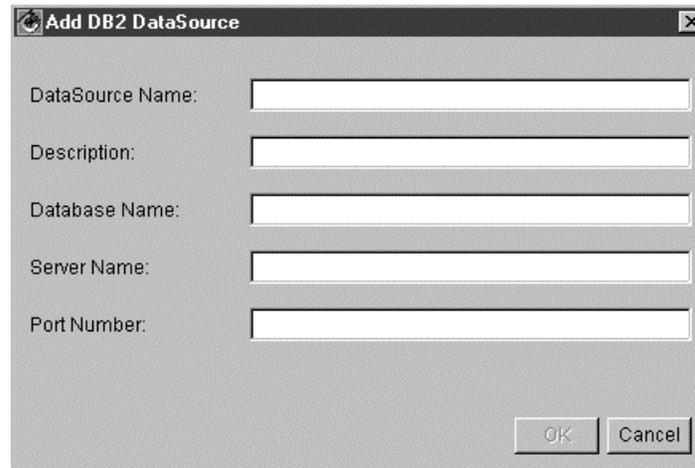
In the Add DB2 DataSource window:

•

  – Type the name of the DataSource in the **DataSource Name** field.
  – Type a description of the DataSource in the **Description** field.
  – Type the name of the database you want to connect to in the **Database Name**
    field.
  – Type the name of the machine where the DB2 server is located in the **Server
    Name** field if you want to use the DB2 net driver
    Com.ibm.db2.jdbc.net.DB2Driver to connect to a database not cataloged
    locally. If you want to use the DB2 app driver
    COM.ibm.db2.jdbc.app.DB2Driver to connect to a database cataloged locally,
    leave the **Server Name** field empty.
  – Type the port number of the DB2 server on the machine where it is located in
    the **Port Number** field if you want to use the DB2 net driver. To use the DB2
    app driver, leave the **Port Number** field empty. If the **Server Name** field
    empty is empty, the **Port Number** field is ignored.
  – Click **OK**. The Add DB2 DataSource window closes, and the Maintain DB2
    DataSources window is displayed. You will see the name of the DataSource
    you just added and the name of the database to which it connects in the
    Maintain DB2 DataSources window.
• Click **Close.** The Connection Alias Definition window is displayed.
• Select the **Auto-commit** check box if you want database updates to be
  automatically committed for each SQL statement. If you do not select the check
  box, database updates are not automatically committed. This check box is
  selected by default.
• Select the **Prompt for logon ID password before connecting** check box if you
  want the user to be prompted for the user ID and password to be used in the
  database connection request. Do not select the check box 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** check box is not selected.
- 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** check box is not selected.

Select **Next** to go to the Advanced page of the Connection Alias window. You can also display the Advanced page by selecting its tab.
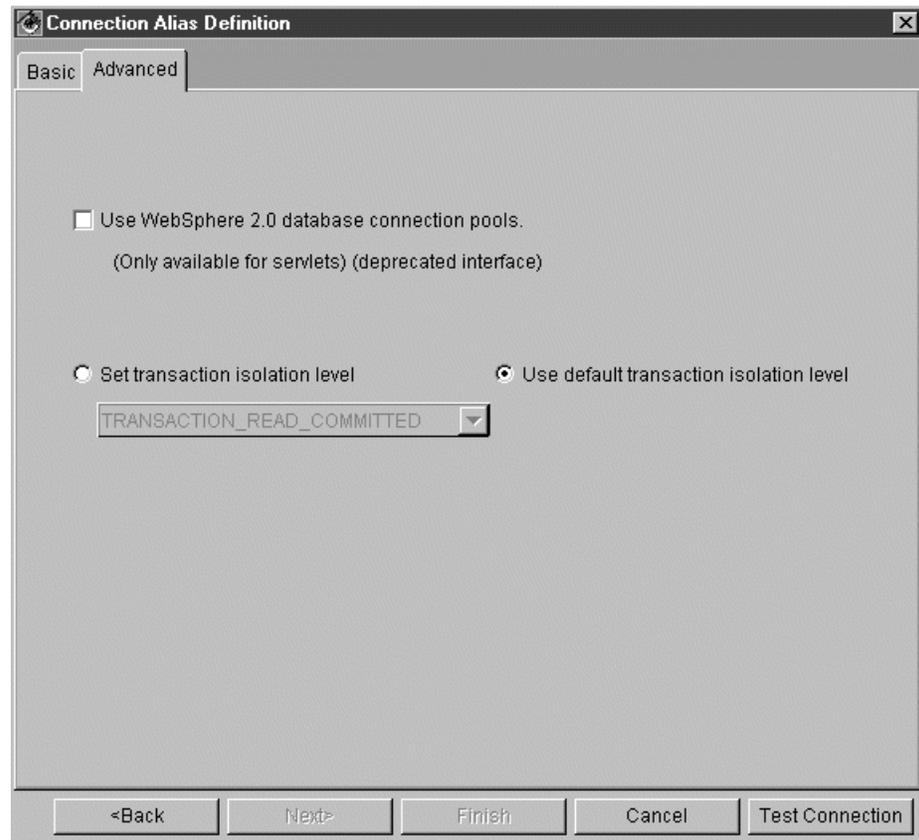
- Select the **Use WebSphere's database connection pools** check box if your application is a servlet that will run in the WebSphere environment and you want to use a connection from the WebSphere connection pool that matches all of the characteristics you specified.

Note for new applications: WebSphere database connection pools have been deprecated and will not be enhanced. Do not use this feature with new applications.
- Click the **Set transaction isolation level** radio button to specify the level of isolation between concurrent transactions. The higher the isolation level, the less access one transaction will have to data that is operated on in another transaction. You can set the transaction isolation level by selecting one of four values:

    TRANSACTION_READ_COMMITED
    TRANSACTION_READ_UNCOMMITTED
    TRANSACTION_REPEATABLE_READ

> TRANSACTION_SERIALIZABLE
>> For information on the transaction isolation level values, refer to the javadoc in java.sql.Connection

If you do not select the **Set the transaction isolation level** radio button, the default transaction isolation level for your database will be used.

- Select **Back** to return to the Basic page of the Connection Alias Definition window. You can also return to the basic page by selecting its tab.
- Click **Test Connection** to test the database connection using the specifications made in the connection alias definition. This test is not sensitive to whether you checked **Use WebSphere's database connection pools**. The connection for the test does not come from a connection pool.
- When you finish defining or editing a connection alias, click **Finish**. Clicking **Finish** 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 data property editor.

RELATED TASKS
Adding resources and paths to the class path

Displaying and navigating a result set

There are a variety of ways in which you may want to display the data in a result set produced by a Select bean or ProcedureCall bean. You could use a set of text fields to display all of the columns in a single row, allowing the user to step through all of the rows one at a time. You could use a JTable to display all of the rows and columns in a tabular form. Or you might want to use various interface components to display only a subset of the result data, such as using a list box to display the values of a single column in all of the rows. Convenient ways of displaying such subsets are provided by a set of beans called Selectors

Row-wise Display and navigation of a result set

To display result set data one row at a time, you can make use of two bound properties that VisualAge for Java generates for each data column in the result set when you use the SQL Assist SmartGuide to compose your SQL statement. 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. The text field will display the value of the column in the current row of the current result set.

For a ProcedureCall bean, these bound properties are only generated if you use the SQL Assist Smart Guide not only to compose the SQL procedure call statement, but also to describe its result sets. The bound properties only appear in the Visual Composition Editor as connectable features of the ProcedureCall if there is exactly one result set described. (This is appropriate either if the stored procedure returns only one result set, or if all of the result sets it returns have the same column structure.)

Even if the bound properties do not appear as connectable features, you can still listen for the events by making an event-to-code connection between the propertyChange event of the ProcedureCall bean and a method. In the method you can check for properties whose names match the syntax Result$n$_columnName or Result$n$_columnName_String, where n is the number of a result set and *columnName*

is the name of a column. When one of these names is found, the method can get the new column value from the ProcedureCall bean and set a property of a visual component to display the value.

To display all of the rows in a result set, you will need to step through its rows. You can accomplish this with the DBNavigator bean. Using the DBNavigator bean, you can set the currentRow property of the associated Select or ProcedureCall bean to:

- the first row in the result set
- the last row in the result set
- the next row in the result set
- 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 or ProcedureCall bean. The DBNavigator bean is designed primarily for use with a Select bean or with a ProcedureCall bean that only returns one result set. It does not allow you to change the currentResult property of an associated ProcedureCall bean. If you need to do this, you must incorporate components in your application for doing so, such as a button with an event-to-method connection to the nextResult method of the ProcedureCall bean.

Tabular display and navigation of a result set

To display result set data in tabular form, you can make a property-to-property connection between the this property of a Select bean or ProcedureCall bean and the model property of a JTable. The JTable will display all of the columns in all of the rows in the cache for the current result set.

Many of the Select bean and ProcedureCall bean methods are designed to operate on the current row of the current result set. Since a JTable has its own row selection mechanisms, you will probably wish to use these mechanisms instead of an associated DBNavigator bean to set the currentRow property of a Select or ProcedureCall bean. (However, you may still wish to use a DBNavigator without its navigation buttons to perform operations such as executing the SQL statement, inserting and deleting rows, and committing changes to the database.)

To use the JTable to set the currentRow property of a Select or ProcedureCall bean you must make a property-to-property connection between the selectedRow property of the JTable and the currentRow property of the Select bean or ProcedureCall bean. Because the selectedRow property is not bound, you will also need to specify, in the connection properties, an event such as mouseClicked, to trigger the propagation of the selectedRow value to the currentRow value . This will insure that whenever you select a new row in the JTable, the corresponding row in the Select bean or ProcedureCall bean will become its current row. This is necessary to insure that methods which operate on the current row (such as UpdateRow and DeleteRow) function as expected.

In additon, if you will be using methods of the Select or ProcedureCall bean that change its currentRow property, such as deleteRow, you need to cause the JTable to reflect these changes. To do this, make an event-to-method connection between the currentRow event of your Select bean or ProcedureCall bean and the setRowSelectionInterval method of the JTable.

If you are using the maximumPacketsInCache property of your Select or ProcedureCall bean to limit the number of rows in the cache, the values of currentRow and currentRowInCache may be different. In this case, since the JTable displays the rows in the cache, you should make all of the above connections to the currentRowInCache property instead of the currentRow property.

If you associate a JTable with a ProcedureCall bean that has multiple result sets, the JTable will always display the current result set. To change the currentResult property of the ProcedureCall bean, you must incorporate components in your application for doing so, such as a button with an event-to-method connection to the nextResult method of the ProcedureCall bean.

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

RELATED TASKS
Connecting beans
Editing Select bean properties
Editing ProcedureCall bean properties
Adding the DBNavigator bean to the Visual Composition Editor surface
Inserting, updating, or deleting data in a result set
Using Selector beans

RELATED REFERENCES
"Select (Database)" on page 75
ProcedureCall
"DBNavigator (Database)" on page 86

Adding Selector beans to the Visual Composition Editor

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

These are nonvisual beans that you use to navigate result sets returned by a Select or ProcedureCall bean. Start by adding one of the beans 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 for a CellSelector, for a RowSelector bean, for a

   ColumnSelector bean, or for a CellRangeSelector bean..
3. Move the mouse pointer to the location on the Visual Composition Editor surface where you want to place the 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 bean is placed at the location of the mouse pointer.

RELATED CONCEPTS
Beans Palette

RELATED TASKS
Adding a feature to VisualAge for Java
Composing beans Visually

Editing CellSelector bean properties

You specify and control the operation of the CellSelector bean by opening the property sheet for the CellSelector bean and setting the values for the displayed properties. To open the property sheet of a CellSelector bean on the Visual Composition Editor surface:

1. Right click the **CellSelector** bean.   The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

You can control:

- The name of the CellSelector bean. Specify the name as the **beanName** property value.
- The name of the column to be selected from the source TableModel. The match on column name is not case sensitive. Specify this in the **columnName** property value. Check the **Show Expert Features** checkbox to display this property. If the specified column name is blank or null, the selected column will be identified by columnNumber.
- The index of the column to be selected from the source TableModel. This column index will only be used if the value of columnName is blank or null. Specify this in the **columnNumber** property value.
- Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. Specify this in the **indexFromOne** property value. Check the **Show Expert Features** checkbox to display this property.
- Which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate propertyChange

events for all these properties. So the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated. Specify this in the **notificationType** property value. Check the **Show Expert Features** checkbox to display this property.

- The index of the row to be selected from the source TableModel. Specify this in the **rowNumber** property value.

Editing RowSelector bean properties

You specify and control the operation of the RowSelector bean by opening the property sheet for the RowSelector bean and setting the values for the displayed properties. To open the property sheet of a RowSelector bean on the Visual Composition Editor surface:

1. Right click the **RowSelector** bean.   The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

You can control:

- The name of the RowSelector bean. Specify the name as the **beanName** property value.
- Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. Specify this in the **indexFromOne** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of columns to be selected. You can select NO_MAXIMUM to indicate that the last column of the selection should be the last column in the source TableModel. Specify this in the **maximumColumns** property value.

- Which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate propertyChange events for all these properties. So the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated. Specify this in the **notificationType** property value. Check the **Show Expert Features** checkbox to display this property.
- The index of the row to be selected from the source TableModel. Specify this in the **rowNumber** property value.
- The index of the column to start the selection from the source TableModel. Specify this in the **startColumnNumber** property value.
- The data type that should be used when populating vectors in response to a vector property query. Some beans may require data to be fed to them in a vector, and may further expect the elements in the vector to be of a specific type. Where this is the case the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property. Specify this in the **vectorContentType** property value. Check the **Show Expert Features** checkbox to display this property.

RELATED CONCEPTS
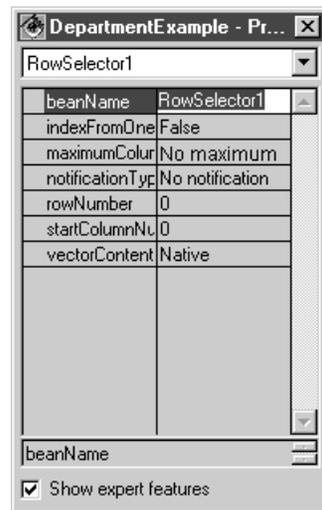Selector beans

RELATED TASKS
Using Selector beans

RELATED REFERENCES
RowSelector

Editing ColumnSelector bean properties

You specify and control the operation of the CellSelector bean by opening the property sheet for the ColumnSelector bean and setting the values for the displayed properties. To open the property sheet of a ColumnSelector bean on the Visual Composition Editor surface:

1. Right click the **ColumnSelector** bean. The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

You can control:

- The name of the ColumnSelector bean. Specify the name as the **beanName** property value.

- The name of the column to be selected from the source TableModel. The match on column name is not case sensitive. Specify this in the **columnName** property value. Check the **Show Expert Features** checkbox to display this property. If the specified column name is blank or null, the selected column will be identified by columnNumber.

- The index of the column to be selected from the source TableModel. This column index will only be used if the value of columnName is blank or null. Specify this in the **columnNumber** property value.

- Whether the column name is to be included as the first element of column data. This only has effect when column data are retrieved as String values. Specify this in the **includeColumnName** property value. Check the **Show Expert Features** checkbox to display this property.

- Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. Specify this in the **indexFromOne** property value. Check the **Show Expert Features** checkbox to display this property.

- The maximum number of rows to be selected. You can select NO_MAXIMUM to indicate that the last row of the selection should be the last row in the source TableModel. Specify this in the **maximumRows** property value.

- Which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate propertyChange events for all these properties. So the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated. Specify this in the **notificationType** property value. Check the **Show Expert Features** checkbox to display this property.

- The index of the row to start the selection from the source TableModel. Specify this in the **startRowNumber** property value.

- The data type that should be used when populating vectors in response to a vector property query. Some beans may require data to be fed to them in a

vector, and may further expect the elements in the vector to be of a specific type. Where this is the case the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property. Specify this in the **vectorContentType** property value. Check the **Show Expert Features** checkbox to display this property.

RELATED CONCEPTS
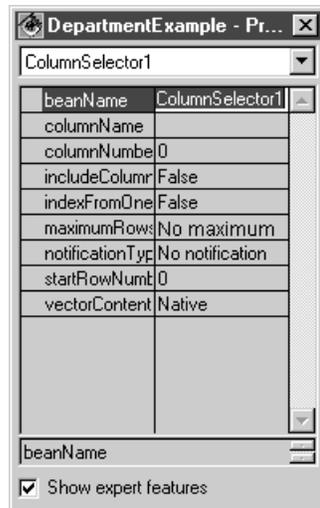Selector Beans

RELATED TASKS
Using Selector beans

RELATED REFERENCES
ColumnSelector

Editing CellRangeSelector bean properties

You specify and control the operation of the CellRangeSelector bean by opening the property sheet for the CellRangeSelector bean and setting the values for the displayed properties. To open the property sheet of a CellRangeSelector bean on the Visual Composition Editor surface:

1. Right click the **CellRangeSelector** bean. The bean menu displays.
2. Click **Properties**. The Properties sheet displays.

You can control:

- The name of the CellRangeSelector bean. Specify the name as the **beanName** property value.
- Whether the column name is to be included as the first element of column data. This only has effect when column data are retrieved as String values. Specify this in the **includeColumnName** property value. Check the **Show Expert Features** checkbox to display this property.
- Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. Specify this in the **indexFromOne** property value. Check the **Show Expert Features** checkbox to display this property.

- Whether the data values are to be inverted row against column during retrieval from, or update to, the source TableModel. The default orientation is row biased. Two dimensional arrays will have one first level dimension for each row, and one second level dimension for each column. When data is inverted then this arrangement will be reversed. Specify this in the **invertData** property value. Check the **Show Expert Features** checkbox to display this property.
- The maximum number of columns to be selected. You can select NO_MAXIMUM to indicate that the last column of the selection should be the last column in the source TableModel. Specify this in the **maximumColumns** property value.
- The maximum number of rows to be selected. You can select NO_MAXIMUM to indicate that the last row of the selection should be the last row in the source TableModel. Specify this in the **maximumRows** property value.
- Which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate propertyChange events for all these properties. So the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated. Specify this in the **notificationType** property value. Check the **Show Expert Features** checkbox to display this property.
- The index of the column to start the selection from the source TableModel. Specify this in the **startColumnNumber** property value.
- The index of the row to start the selection from the source TableModel. Specify this in the **startRowNumber** property value.
- The data type that should be used when populating vectors in response to a vector property query. Some beans may require data to be fed to them in a vector, and may further expect the elements in the vector to be of a specific type. Where this is the case the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property. Specify this in the **vectorContentType** property value. Check the **Show Expert Features** checkbox to display this property.

RELATED CONCEPTS
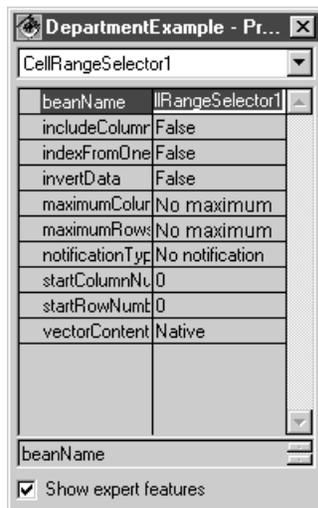Selector beans

RELATED TASKS
Using Selector beans

RELATED REFERENCES
CellRangeSelector

Using Selector Beans

Selector beans operate on a Java TableModel. Used with the Select bean you can query a database, receive a result set back and use the selector beans to process a subset of the result set. The ColumnSelector and RowSelector beans implement the ComboBoxModel which means they can extract data from a result set and pass it to a user interface such as a JList. You can also use them to get data from a user interface, put the data in a result set and then have a Select or ProcedureCall bean update the database.

Selector beans can also be used to work with arrays. For example, you could use the RowSelector bean to extract a row from a result set, do data conversion on the row, and pass the result as an array.

All selectors have properties that define the subset of TableModel data that they are operating on. For a CellSelector these identify the row and column indices that define the location of the cell within the source TableModel. For the other selectors, similar properties define the start row and/or column indices, and the maximum number of rows and/or columns to be selected.

The above properties are by default zero-based. That is, the first row or column is deemed to be row or column zero. Many navigation Java beans have a value property that is also zero-based, but in cases where a navigator value is one-based the selectors have a boolean property that can switch them to one-based operation.

These properties may be set at design-time in a visual builder, and/or modified at run-time by connecting them to visual Java beans that provide navigation, such a JList.

Data Conversion

To provide the maximum opportunity for connecting selectors to other Java beans, all selectors will attempt data conversion on request. For example, the source data might be numeric, but held in a character representation as a string. In this circumstance a selector will be able to return the value in any one of the numeric Java types, and accept values for update with the same flexibility. Data conversion will not always be possible, and numeric overflow might occur when converting numbers to forms using less memory. However, in many cases conversion will enable a connection that might not otherwise be possible.

Conversion is achieved using the data access properties which are available on all selectors. These properties are named after the data type that they return and accept, and have "getter" and "setter" methods. For example: `int getInt()` and `void setInt(int value);`

Data Changes

If the source TableModel for a selector changes, the selector will usually notify its listeners of the data change. But if rows are inserted or deleted prior to the rows that have been selected, the selector will adjust its rowNumber/startRowNumber property so that the rows selected remain the same.

There are a number of ways in which interested Java beans may be informed of changes to data currently targeted by a selector:

- **SelectorListener**
  All selectors support listeners implementing the SelectorListener interface. These listeners will be notified whenever source data covered by the selector, or the selection criteria for the selector, are modified.

  On change notification, listeners may retrieve the updated data using any of the available data accessors, provided that coercion between the source data type and the data type of the accessor is possible.

- **ListDataListener**
  RowSelector and ColumnSelector implement the ComboBoxModel interface and support listeners implementing the ListDataListener interface. These listeners

will be notified whenever the source data covered by the selector, or the selection criteria for the selector, are modified.

On change notification, data will normally be retrieved using the ComboBoxModel services supported by these selectors.

- **TableModelListener**

CellRangeSelector implements the TableModel interface, and support listeners implementing the TableModelListener interface. These listeners will be notified whenever the source data covered by the selector, or the selection criteria for the selector, are modified.

On change notification, data will normally be retrieved using the TableModel services supported by this selector.

- **PropertyChangeListener**

All selectors have bound properties and support listeners implementing the PropertyChangeListener interface. Property changes are notified for most selector properties, including the data access properties.

If source data covered by the selector, or the selection criteria for the selector, are modified, then in theory all the data access properties might be deemed to have changed. In practice, notifying a property change event for all of that data access properties would be counter productive, since most listeners will only be interested in data of one type. However, some visual builders support the explicit binding of named properties, and to support these it is necessary to signal property change events. To allow for this without having to incur the overhead of many property change notifications for every data change, the NotificationType property may be customized to define which of the data access properties on a selector will be notified as changed.

RELATED CONCEPTS
Selector Beans

RELATED TASKS
Editing CellSelector Bean Properties
Editing ColumnSelector Bean Properties
Editing RowSelector Bean Properties
Editing CellRangeSelector Bean Properties
Using Selector Bean Data Access Properties

Using Selector bean data access properties

Selector bean data access properties allow you to retrieve values from the underlying TableModel and perform automatic data conversion on the values. Depending on the selector used, the value(s) are returned as a single value (CellSelector), an 1-dimensional array of values (ColumnSelector and RowSelector), or a 2-dimensional array of values(CellRangeSelector).

The data access properties that are available on the Connectable Features dialog for all selectors are:

- BigDecimal - value(s) are java.math.BigDecimal data types
- BigInteger - value(s) are java.math.BigInteger data types
- boolean - value(s) are boolean data types
- byte - value(s) are byte data types
- byte[] - value(s) are arrays of byte data type
- Date - value(s) are java.sql.Date data types
- double - value(s) are double data types

- float - value(s) are float data types
- int - value(s) are int data types
- long - value(s) are long data types
- Object - value(s) are Object data types
- short - value(s) are short data types
- String - value(s) are String data types
- Time - value(s) are java.sql.Time data types
- Timestamp - value(s) are java.sql.Timestamp data types
- Vector - value(s) are Vector data types (not availabe for CellSelector)
- VectorOfVectors - value is a vector of Vector data types (CellRangeSelector only)

All of these properties are expert features. You must check the **Show Expert Features** checkbox on the Connectable Features dialog to display them.

All of the data access properties are defined as bound properties. However, when the underlying data is changed, propertyChange events will not automatically be triggered for all of these properties. A propertyChange event will only be triggered for at most one data access property for a selector. The notificationType property of the selector controls which propertyChange event will be triggered. By default, this property is set to Selector.NONE which means no propertyChange event will be triggered. If you want a propertyChange event to be triggered, you must set this property to an appropriate value based on what propertyChange event you want to be triggered. For example, if you have made a connection using the String data access property, you would set the notificationType to String.

If you use the Vector data access property, you can control the data type of the elements in the vector through the vectorContent property of a Selector.

RELATED CONCEPTS
Selector Beans

RELATED TASKS
Using Selector beans

RELATED REFERENCES
CellSelector
RowSelector
ColumnSelector
CellRangeSelector

Inserting, updating, or deleting data in a result set

The Select and ProcedureCall beans provide methods that you can use to insert, update, and delete rows in a result set. To perform these operations you must first use a Select or ProcedureCall bean to retrieve one or more result sets. (A ProcedureCall bean can retrieve more than one result set at a time while a Select bean has only one result set.) You then apply the changes to the current row of the current result set.

Inserting data

Inserting data via a Select or ProcedureCall bean is a 3-step process. You must:

1. Add a new empty row to the current result set.

2. Set one or more values in the empty row.
3. Move to another row or use the updateRow method to trigger insertion of the new row into the database.

By coding a method

One way of accomplishing these tasks is to make an event-to-code connection between an appropriate interface component, such as a button, and a method. For example, you can make an event-to-code connection between the actionPerformed event for a button and a method which executes methods of the Select or ProcedureCall bean. In this method, include code that calls the newRow method to add the empty row, the setColumnValue method to provide one or more values, and the updateRow or any method that moves to a different row to trigger insertion of the new row into the database.

Without coding a method

If you prefer not to write a method, you can use connections to a DBNavigator bean and to an interface component where your data is displayed to accomplish the same thing.

To use the DBNavigator bean, create a property-to-property connection between the this property of the Select bean or ProcedureCall bean and the model property

of a DBNavigator bean. The DBNavigator bean has an Insert button that adds a new empty row to the current result set.

If your data is displayed in a JTable, when you enter a new value into a cell of the table, the value is automatically set in the corresponding row and column of the result set.

If your data is displayed one row at a time in other interface components, such as text fields, which you have connected to the bound column properties of the Select bean or ProcedureCall bean, you need to insure that any value you provide in the interface component gets propagated to the corresponding row and column of the result set. If the property of the interface component is bound, VisualAge for Java generates code to do the propagation. However, many properties of interface components, including the text property of a text field, are not bound. In this case, you must specify in the connection properties, an event to trigger the propagation of the updated value to the column property.

Finally, you can use the Next button of the DBNavigator bean (or any other button on the DBNavigator that moves to another row) to trigger insertion of the new row into the database. If your data is displayed in a JTable and you are using the row selection mechanisms of the JTable to control the current row of the result set you can click on another row to trigger insertion of the new row. Or you can make an event-to-method connection between another interface component, such as a button, and the updateRow method of the Select bean or ProcedureCall bean.

Updating data

Updating data via a Select or ProcedureCall bean is a 2-step process. You must:
1. Set one or more values in the current row of the current result set.

2. Move to another row or use the UpdateRow method to trigger the update of the row in the database.

By coding a method

One way of accomplishing these tasks is to make an event-to-code connection between an appropriate interface component, such as a button, and a method. For example, you can make an event-to-code connection between the actionPerformed event for a button and a method which executes methods of the Select bean or ProcedureCall bean. In this method, include code that calls the setColumnValue method to provide one or more values, and updateRow or any method that moves to a different row to trigger the update of the row in the database.

Without coding a method

If you prefer not to write a method, you can use connections to a DBNavigator bean and to an interface component where your data is displayed to accomplish the same thing.

If your data is displayed in a JTable, when you enter a new value into any cell of the table, the value is automatically set in the corresponding row and column of the result set.

If your data is displayed one row at a time in other interface components, such as text fields, which you have connected to the bound column properties of the Select bean or ProcedureCall bean, you need to insure that any value you provide in the interface component gets propagated to the corresponding row and column of the result set. If the property of the interface component is bound, VisualAge for Java generates code to do the propagation. However, many properties of interface components, including the text property of a text field, are not bound. In this case, you must specify in the connection properties, an event to trigger the propagation of the updated value to the column property.

Once your values have been set, you can use the Next button of an associated DBNavigator bean (or any other button on the DBNavigator that moves to another row) to trigger the update of the row in the database. If your data is displayed in a JTable and you are using the row selection mechanisms of the JTable to control the current row of the result set you can click on another row to trigger the update of the new row in the database. Or you can make an event-to-method connection between another interface component, such as a button, and the updateRow method of the Select bean or ProcedureCall bean.

Deleting data

Deleting data via a Select or ProcedureCall bean is a 1-step process. You need only use the deleteRow method to delete the current row from the database.

The Delete button of an associated DBNavigator bean executes the deleteRow method. If you are not using a DBNavigator with a Delete button, you can make an event-to-method connection between another interface component, such as a button, and the deleteRow method of the Select bean or ProcedureCall bean.

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

Adding the DBNavigator bean to the Visual Composition Editor surface

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

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 CONCEPTS
Beans Palette
RELATED TASKS
Composing beans visually

Starting the Create Database Application SmartGuide

Before you start the Create Database Application SmartGuide, you must add the Data Access Beans feature to your workspace. To add the Data Access beans, select **File > Quick Start > Features > Add Features**; then select **Data Access Beans 3.0** and click **OK**.

You can start the Create Database Application SmartGuide in one of these ways:

• Select **File > Quick Start > Basic > Create Database Application** and click **OK**.

• In the Workbench window, right-click the project or package for which you want to create a database application; then, select **Tools > Create Database Application**.

- In the Workbench window, select the project or package for which you want to create a database application; then, select **Selected > Tools > Create Database Application** from the Workbench menu.

If you choose to start the Create Database Application SmartGuide by using the Quick Start window from the File menu, Visual Age for Java does not enter a project name and package name in the SmartGuide. You must enter this information yourself in the SmartGuide.

RELATED CONCEPTS
About the Create Database Application SmartGuide

RELATED TASKS
Creating a database application with the Create Database Application SmartGuide

Creating a database application with the Create Database Application SmartGuide

Requirement: Before you can use the Create Database Application SmartGuide, you must create a package for the data access class.

To create a database application with the Create Database Application SmartGuide, perform these tasks:

1. Start the Create Database Application SmartGuide.
2. In the Create Database Application page, enter a project name, package name, and class name for the application. If you started the SmartGuide from a selected project or package in the Workbench window, the project name or package name is already entered for you. Click **Next**. The Database Connection and SQL Specification page appears.
3. Specify a database connection alias. (page 70)
4. Create an SQL specification. (page 70)
5. Create a GUI for the database application. (page 71)
6. Save your database application by selecting **Bean > Save Bean** in the Visual Composition Editor window.

Specifying a database connection alias for a database application
1. In the Database Connection and SQL Specification page, click **Edit**. The Query Editor appears.
2. In the Query Editor, select the Connection tab. Select an existing connection alias and click **OK**.
   or
   Create a new connection alias by clicking **Add**. The Connection Alias Definition window appears.
   See Specifying a connection alias and Defining or editing a connection alias.
3. Once you have specified a database connection alias, click **OK** in the Query Editor. the Database & SQL Specification windows appears.

Creating an SQL specification for a database application
1. In the Database & SQL Specification windows, Click **Edit**. The Query Editor appears.
2. If you have more than one data access class, verify or select the data access class in the **Database Access Class** field for the database connection alias. The

data access class selected on the Connection page is not automatically pre-selected in the SQL page of the Query Editor.

3. In the Query Editor, select the SQL tab. Select an existing SQL specification and click **OK**.
   or
   Create an SQL specification by clicking **Add**. The New SQL Specification window appears.

4. When creating an SQL specification, select **Use SQL Assist SmartGuide** in the New SQL Specification window. Using the SQL Assist SmartGuide is the recommended method for create an SQL specification. Click **OK**. The SQL Assist SmartGuide appears.
   See Composing an SQL query.

5. Once you have created your SQL statement, click **OK** in the Query Editor. The Database Connection & SQL Specification page appears. The page shows the names of the connection alias and the SQL specification that will be used to initialize the query property of a Select bean.

6. **Optional**: Click **Edit** to return to the Query Editor and modify your SQL. When you have finished editing your SQL statement, click **OK** in the Query Editor. The Database Connection & SQL Specification page appears.

7. Click **Next**. The Database View Specification window appears.

Creating a GUI for a database application

1. In the Database View Specification window, define a user interface by selecting the elements that you want to use for the database application.
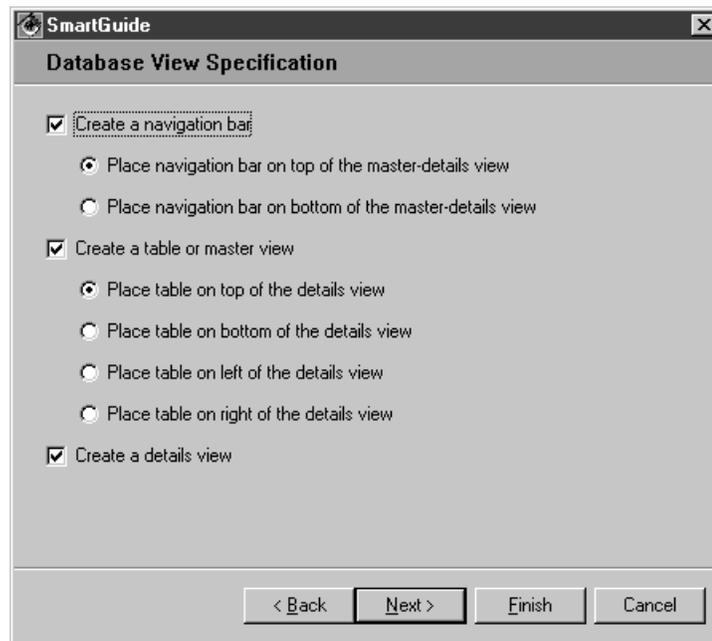
2. Click **Next**. The Details View Specification window appears. For the current row, the details view displays the values for each column in separate text fields and labels each text field with the name of the column.
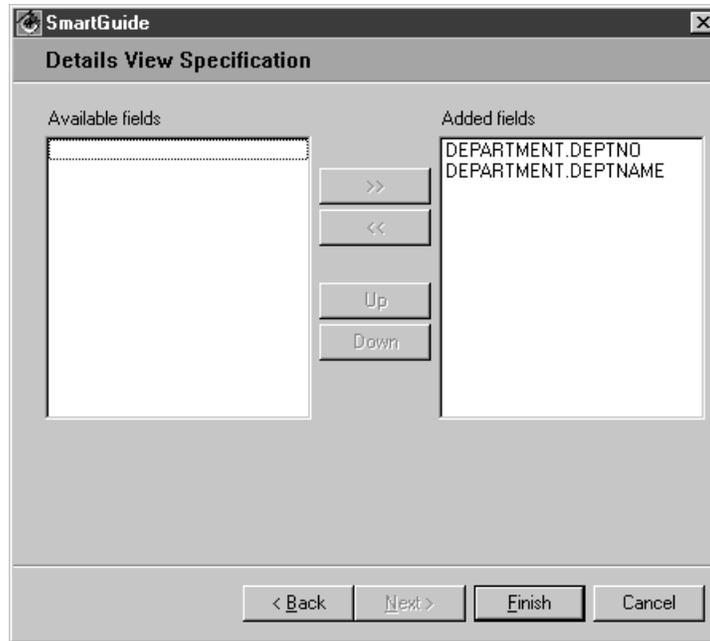
3. **Optional**: In the Details View Specification window, select fields from the **Added fields** list and click **<<** to remove them from the database application GUI. The fields are moved to the **Available fields** list. The fields in the **Available fields** list will be part of your SQL query and will appear in the JTable view. However, the fields will not appear in the Details view.

   Note: If you did *not* use the SQL Assist SmartGuide to create your SQL statement, the **Added fields** list is blank because the Create Database Application SmartGuide does not have information about the columns in the Select bean to generate the Details view.

4. **Optional**: Select a field and click **Up** or **Down** to reorder the fields in your GUI.

5. Click **Finish**. The database application appears in the Visual Composition Editor of the IDE.

6. Use the Visual Composition Editor to modify the generated database application GUI.
   See Visual Composition Editor overview.
   For information about updating, inserting, or deleting rows, see Inserting, updating, or deleting data in a result set.

   The Visual Composition Editor shows the method updateTableSelection() for the Select class bean. This method synchronizes the selected row in the JTable with the current row in the Select bean.
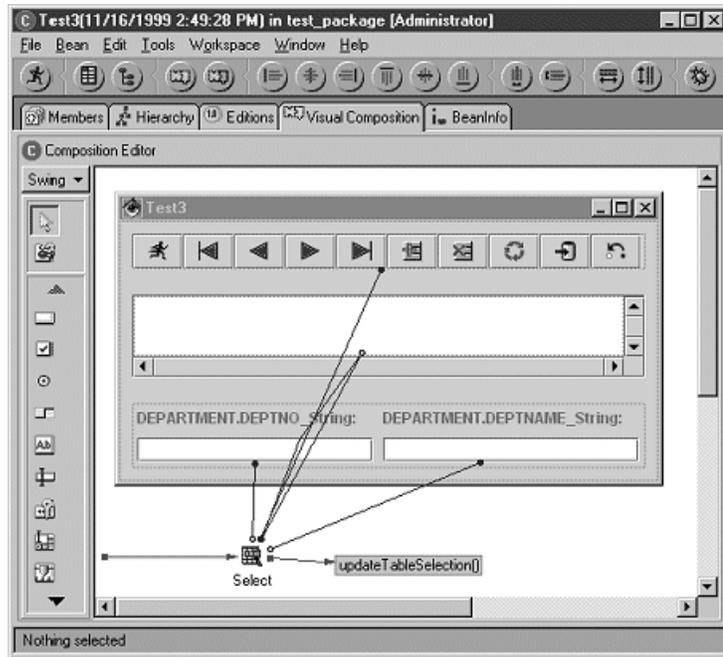
7. If you want your database application to update rows in a database table, add events for the connection to each text field. Right-click a connection to a field and select **Properties**. The Property-to-Property connection Properties window appears. In the **Target event** field, select an event for the field. For example, the actionPerformed event is triggered when a user presses Enter after typing a new value in a text field.

   The connections to a text property of a JText field are not a bound property. Therefore, VisualAge for Java will not generate code to do the propagation. In the connection properties, you must specify an event to trigger the propagation of the updated value to the column property.

   See Property-to-property connections.

RELATED CONCEPTS

About the Create Database Application SmartGuide

Chapter 3. Data Access Beans

The following beans provide access to relational data:

Bean	Description
"Select (Database)"	A non-visual bean used to query relational data
Modify	A non-visual bean used to modify relational data
ProcedureCall	A non-visual bean used to run a database stored procedure
CellSelector	A non-visual bean used to select a cell from a TableModel
ColumnSelector	A non-visual bean used to select a column from a TableModel
RowSelector	A non-visual bean used to select a row from a TableModel
CellRangeSelector	A non-visual bean used to select a range of cells from a TableModel
"DBNavigator (Database)" on page 86	A visual bean that provides navigation used with a Select or ProcedureCall bean

RELATED TASKS
"Chapter 2. Accessing Relational Data" on page 11

Select (Database)

Use a Select bean ![icon] to access relational data.

Select Bean Properties

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

distinctTypeEnabled
> Whether to enable inserts, updates, and deletes for result sets which contain user-defined (distinct) types. The default value is *False*.

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*.

forceSearched
Whether to force generation of searched rather than positioned SQL UPDATE and DELETE statements for the result set returned by the statement. The default value is *False*.

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. The default value is 1.

query Specifies the connection alias and SQL specification for the Select bean. See "Specifying a connection alias" on page 47 and "Making an SQL specification" on page 12 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. The default value is *False*.

timeout
The maximum number of seconds allowed for the statement to execute. The default value is 0, which means no maximum.

validateLOBs
Specifies whether to check the validity of a Clob or Blob object before returning the object to you as a column or parameter value. A commit or rollback may cause LOB objects to become invalid. If a LOB is not valid, it is re-fetched from the database and returned to you. If the attempt to re-fetch the value fails, an exception is thrown.

Note: A re-fetched LOB may have different contents than the original LOB.

RELATED TASKS
"Chapter 2. Accessing Relational Data" on page 11
RELATED REFERENCES

Modify (Database)

Use the Modify bean to run SQL INSERT, UPDATE, or DELETE statements without first running a query and retrieving its result set.

Modify Bean Properties

action Specifies the connection alias and SQL specification for the Modify bean. See "Specifying a connection alias" on page 47 and "Making an SQL specification" on page 12 for further information.

beanName
>Specifies the name of the Modify bean instance. It must follow standard naming rules for beans. The default name is Modifyn, where n is the number of Modify beans with default names; for example, the first default name is Modify1.

timeout
>The maximum number of seconds allowed for the statement to execute. The default value is 0, which means no maximum.

RELATED TASKS
"Chapter 2. Accessing Relational Data" on page 11

RELATED REFERENCES

ProcedureCall (Database)

Use the ProcedureCall bean access relational data.

ProcedureCall Bean Properties

beanName
>Specifies the name of the ProcedureCall bean instance. It must follow standard naming rules for beans. The default name is ProcedureCalln, where n is the number of ProcedureCall beans with default names; for example, the first default name is ProcedureCall1.

distinctTypeEnabled
>Whether to enable inserts, updates, and deletes for result sets which contain user-defined (distinct) types. The default value is *False*.

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.

fillResultCacheOnExecute
Whether as many result sets as possible should be fetched into memory as soon as you execute your stored procedure or result sets should be fetched only as you ask for them. The default value is *True.*

forceSearched
Whether to force generation of searched rather than positioned SQL UPDATE and DELETE statements for the result set returned by the statement. The default value is *False.*

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. The default value is 1.

procedure
Specifies the connection alias and SQL specification for the ProcedureCall bean. See "Specifying a connection alias" on page 47 and "Making an SQL specification" on page 12 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. The default value is *False.*

timeout
The maximum number of seconds allowed for the statement to execute. The default value is 0, which means no maximum.

validateLOBs
Specifies whether to check the validity of a Clob or Blob object before returning the object to you as a column or parameter value. A commit or rollback may cause LOB objects to become invalid. If a LOB is not valid, it is re-fetched from the database and returned to you. If the attempt to re-fetch the value fails, an exception is thrown.

Note: A re-fetched LOB may have different contents than the original LOB.

CellSelector (Database)

Use the CellSelector [icon] to view a cell in a TableModel such as a result set returned by a Select or ProcedureCall bean.

CellSelector Properties

beanName
> Specifies the name of the CellSelector bean instance. It must follow standard naming rules for beans. The default name is CellSelector$n$, where n is the number of CellSelector beans with default names; for example, the first default name is CellSelector1.

columnName
> Specifies the name of the column selected from the source TabelModel. The match on column name is not case sensitive. If the specified column name is blank or null, the selected column will be identified by columnNumber.

columnNames
> An array containing the names of the columns that are currently selected.

columnNumber
> The index of the column to be selected from the source TableModel. This column index will only be used if the value of columnName is null or blank string.

indexFromOne
> Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. The default is false.

model The Java TableModel used as the source of the data for the selection.

notificationType
> Specifies which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate property change events for all these properties, so the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated.

rowNumber
> The index of the row to be selected from the source TableModel.

The following are data access properties used by CellSelector bean for data conversion.

BigDecimal
>	Value of the cell as a java.math.BigDecimal data type.

BigInteger
>	Value of the cell as a java.math.BigInteger data type.

boolean
>	Value of the cell as a boolean data type.

byte	Value of the cell as a byte data type.

byte[]	Value of the cell as a byte array data type.

Date	Value of the cell as a java.sql.Date data type.

double
>	Value of the cell as a double data type.

float	Value of the cell as a float data type.

int	Value of the cell as an int data type.

long	Value of the cell as a long data type.

Object
>	Value of the cell as an Object data type.

short	Value of the cell as a short data type.

String	Value of the cell as a String data type.

Time	Value of the cell as a java.sql.Time data type.

Timestamp
>	Value of the cell as a java.sql.Timestamp data type.

RELATED TASKS

Editing CellSelector Bean Properties
Using Selector Bean Data Access Properties

RELATED REFERENCES

CellRangeSelector
ColumnSelector
RowSelector

ColumnSelector(Database)

Use the ColumnSelector bean ![icon] to view a column in a TableModel such as a results set returned by Select or ProcedureCall beans.

CellSelector Properties

beanName
>	Specifies the name of the ColumnSelector bean instance. It must follow standard naming rules for beans. The default name is ColumnSelector*n*, where *n* is the number of ColumnSelector beans with default names; for example, the first default name is ColumnSelector1.

columnName
>	Specifies the name of the column selected from the source TabelModel. The match on column name is not casr sensitive. If the specified column name is blank or null, the selected column will be identified by columnNumber.

columnNames
> An array containing the names of the columns that are currently selected.

columnNumber
> The index of the column to be selected from the source TableModel. This column index will only be used if the value of columnName is null or blank string.

includeColumnName
> Whether the column name is to be included as the first element of column data. This only has effect when column data are retrieved as String values. The default is false.

indexFromOne
> Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. The default is false.

maximumRow
> The maximum number of rows to be selected. You can select NO_MAXIMUM to indicate that the last row of the selection should be the last row in the source TableModel. The default is NO_MAXIMUM.

model The Java TableModel used as the source of the data for the selection.

notificationType
> Specifies which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate property change events for all these properties, so the property of interest may be specified, limiting property change events to a single property.

startRowNumber
> The index of the row to be selected from the source TableModel.

vectorContentType
> The data type used to populate vectors in response to a vector property query. Some beans may require data be fed to them in a vector, and may further expect the elements in the vector to be of a specific type. Where this is the case, the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property.

The following are data access properties used by ColumnSelector bean for data conversion.

BigDecimal
> A 1-dimensional array of java.math.BigDecimal data types.

BigInteger
> A 1-dimensional array of java.math.BigInteger data types.

boolean
> A 1-dimensional array of boolean data types.

byte A 1-dimensional array of byte data types.

byte[] A 1-dimensional array of byte array data types.

Date A 1-dimensional array of java.sql.Date data types.

double
A 1-dimensional array of double data types.

float A 1-dimensional array of float data types.

int A 1-dimensional array of int data types.

long A 1-dimensional array of long data types.

Object
A 1-dimensional array of Object data types.

short A 1-dimensional array of short data types.

String A 1-dimensional array of String data types.

Time A 1-dimensional array of java.sql.Time data types.

Timestamp
A 1-dimensional array of java.sql.Timestamp data types.

Vector A 1-dimensional array of Vector data types. The data type of the elements
of the vector is controlled by the vectorContentType property.

RELATED TASKS
Editing ColumnSelector Bean Properties
Using Selector Bean Data Access Properties

RELATED REFERENCES
CellSelector
CellRangeSelector
RowSelector

RowSelector(Database)

Use the RowSelector bean to view a row in a TableModel such as a result set
returned by a Select or ProcedureCall bean.

RowSelector Bean Properties

beanName
Specifies the name of the RowSelector bean instance. It must follow
standard naming rules for beans. The default name is RowSelector*n*, where
n is the number of RowSelector beans with default names; for example, the
first default name is RowSelector1.

columnNames
An array containing the names of the columns that are currently selected.

indexFromOne
Whether the row and column indexes are assumed to index from one. This
property should be set to true where the selection row and/or column
properties are connected to a bean property that indexes from one. If set to
false, indexing is from zero. The default is false.

maximumColumns
The maximum number of columns to be selected. You can select
NO_MAXIMUM to indicate that the last column of the selection should be
the last column in the source TableModel. The default is NO_MAXIMUM.

model The Java TableModel used as the source of the data for the selection.

notificationType
> Specifies which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate property change events for all these properties, so the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated.

rowNumber
> The index of the row to be selected from the source TableModel.

startColumnNumber
> The index of the column to be selected from the source TableModel.

vectorContentType
> The data type used to populate vectors in response to a vector property query. Some beans may require data be fed to them in a vector, and may further expect the elements in the vector to be of a specific type. Where this is the case, the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property.

The following are data access properties used by RowSelector bean for data conversion.

BigDecimal
> A 1-dimensional array of java.math.BigDecimal data types.

BigInteger
> A 1-dimensional array of java.math.BigInteger data types.

boolean
> A 1-dimensional array of boolean data types.

byte A 1-dimensional array of byte data types.

byte[] A 1-dimensional array of byte array data types.

Date A 1-dimensional array of java.sql.Date data types.

double
> A 1-dimensional array of double data types.

float A 1-dimensional array of float data types.

int A 1-dimensional array of int data types.

long A 1-dimensional array of long data types.

Object
> A 1-dimensional array of Object data types.

short A 1-dimensional array of short data types.

String A 1-dimensional array of String data types.

Time A 1-dimensional array of java.sql.Time data types.

Timestamp
> A 1-dimensional array of java.sql.Timestamp data types.

Vector A 1-dimensional array of Vector data types. The data type of the elements of the vector is controlled by the vectorContentType property.

CellRangeSelector(Database)

Use the CellRangeSelector bean ⊞ to view a 2-dimensional array from a TableModel such as a result set returned by a Select or ProcedureCall bean.

CellSelector Properties

beanName
Specifies the name of the CellRangeSelector bean instance. It must follow standard naming rules for beans. The default name is CellRangeSelector*n*, where *n* is the number of CellRangeSelector beans with default names; for example, the first default name is CellRangeSelector1.

columnNames
An array containing the names of the columns that are currently selected.

includeColumnNames
Whether the column name is to be included as the first element of column data. This only has effect when column data are retrieved as String values. The default is false.

indexFromOne
Whether the row and column indexes are assumed to index from one. This property should be set to true where the selection row and/or column properties are connected to a bean property that indexes from one. If set to false, indexing is from zero. The default is false.

invertData
Whether the data values are to be inverted row against column during retrieval from, or update to, the source TableModel. The default orientation is row biased. Two dimensional arrays will have one first level dimension for each row, and one second level dimension for each column. When data is inverted, then this arrangement will be reversed. The default is false.

maximumColumns
The maximum number of columns to be selected. You can select NO_MAXIMUM to indicate that the last column of the selection should be the last column in the source TableModel. The default is NO_MAXIMUM.

maximumRows
The maximum number of rows to be selected. You can select NO_MAXIMUM to indicate that the last row of the selection should be the last row in the source TableModel. The default is NO_MAXIMUM.

model The Java TableModel used as the source of the data for the selection.

notificationType

Specifies which data access property will be notified of changes by a propertyChange event. Selectors are able to convert source data into a variety of data types, each of which has a corresponding bound property. All of these properties are theoretically altered when the selector data source or selection criteria are changed. However, it would be very inefficient to generate property change events for all these properties, so the property of interest may be specified, limiting property change events to a single property. By default, no property change events are generated.

startColumnNumber

The index of the column to start the selection from the source TableModel.

startRowNumber

The index of the row to start the selection from the source TableModel.

vectorContentType

The data type used to populate vectors in response to a vector property query. Some beans may require data be fed to them in a vector, and may further expect the elements in the vector to be of a specific type. Where this is the case, the notificationType property may be set to Selector.VECTOR to cause propertyChange events to be generated for the vector property.

The following are data access properties used by the CellRangeSelector bean for data conversion.

BigDecimal

A 2-dimensional array of java.math.BigDecimal data types.

BigInteger

A 2-dimensional array of java.math.BigInteger data types.

boolean

A 2-dimensional array of boolean data types.

byte A 2-dimensional array of byte data types.

byte[] A 2-dimensional array of byte array data types.

Date A 2-dimensional array of java.sql.Date data types.

double

A 2-dimensional array of double data types.

float A 2-dimensional array of float data types.

int A 2-dimensional array of int data types.

long A 2-dimensional array of long data types.

Object

A 2-dimensional array of Object data types.

short A 2-dimensional array of short data types.

String A 2-dimensional array of String data types.

Time A 2-dimensional array of java.sql.Time data types.

Timestamp

A 2-dimensional array of java.sql.Timestamp data types.

Vector[]
> A 1-dimensional array of Vector data types. The data type of the elements of the vector is controlled by the vectorContentType property.

VectorOfVectors
> A vector of Vector data types. The data type of the elements of the vector is controlled by the vectorContentType property.

RELATED TASKS
Editing CellRangeSelector Bean Properties
Using Selector Bean Data Access Properties

RELATED REFERENCES
CellSelector
ColumnSelector
RowSelector

DBNavigator (Database)

Use the DBNavigator bean [image] with a Select or ProcedureCall bean to access relational data. The DBNavigator bean provides a set of Buttons (page 87) that execute the SQL statement for the associated 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.

DBNavigator Bean Properties

beanName
> Specifies the name of the DBNavigator bean instance. It must follow standard naming rules for beans. The default name is DBNavigator*n*, 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. Retrieves the latest information from the database, using the SQL statement for the associated Select Bean and the current database connection. 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 2. Accessing Relational Data" on page 11
RELATED REFERENCES
"Select (Database)" on page 75
ProcedureCall

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1997, 2000. All rights reserved.

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

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.