INTRODUCTION TO DATABASES

One of the most important features of JavaServer Pages technology is the ability to connect to a database. Databases store and efficiently manage large collections of information. JSP pages can be used to make this information available to the users who visit your Web site or to store information submitted by users.

Instead of storing information in text files or static Web pages, a JSP page can be set up to retrieve, format and display data from a database. When a user accesses the JSP page, the information displayed by the page will be created from the current information in the database. A JSP page can also allow users to manipulate the data in a database.

Using databases to store information and using JSP pages to access the information is an efficient method of displaying up-to-date information in a Web site.

DATABASE PROGRAMS

There are several different programs available that you can use to create a database. The two most popular database programs used when working with Windows-based systems are Microsoft Access and Microsoft SQL Server. Microsoft Access is useful for creating relatively small databases, while Microsoft SQL Server is useful for creating large databases, such as a database used to provide information to a busy e-commerce Web site.

For information about Microsoft Access and Microsoft SQL Server, you can visit the www.microsoft.com/office/access and www.microsoft.com/sql Web sites.

Two popular database programs used when working with UNIX-based systems are MySQL and PostgreSQL. Information about these database programs is available at the www.mysql.com and www.postgresql.org Web sites.

DATABASE STRUCTURE

A database is made up of one or more tables. A table contains records that store the information entered into the table. For example, a record could store the information for one customer. Each record is divided into fields. A field is a specific piece of information in a record, such as the first name of a customer.

Great care should be taken when initially planning and designing the structure of a database. A well-planned database ensures that tasks, such as adding or deleting records, can be performed efficiently and accurately. Poor database design may cause problems if the database needs to be changed in the future.

CONNECT TO A DATABASE

Before a JSP page can access a database, you must create a connection to the database. On Windows-based systems, you can first create a Data Source Name (DSN) for the database to tell your JSP pages what kind of database you want to connect to and where the database is located. You can then use the DSN with the `java.sql` package in a JSP page to connect the page to the database.

Once connected, you can easily access the database to add, modify and delete records, as well as administer the database.

STRUCTURED QUERY LANGUAGE

In order for a JSP page to work with the records in a database, the page must be able to communicate with the database. You use the Structured Query Language (SQL) in a JSP page you want to communicate with a database.

SQL FEATURES

Standardized

SQL is the industry standard language for managing and manipulating data in a database. SQL can be used to work with many types of databases, which makes it easy to upgrade from one database program to another. For example, a small Web site might start out using a Microsoft Access database, but then grow large enough to require a database created using Microsoft SQL Server. You need to learn only one language to have your JSP pages communicate with both types of databases.

Easy to Use

SQL is a very simple language to work with and uses many easy-to-understand commands. For example, SQL uses the `INSERT` statement to add information to a database. These plain-language commands make it easy for you to read code created using SQL and determine the purpose of the code.

Powerful

Although SQL is easy to use, it is a very powerful language. As well as being suitable for retrieving data from a database and performing simple tasks such as adding and deleting records, SQL can be used to perform complicated procedures, such as compiling different types of data from multiple data sources.

SQL STATEMENTS

Although SQL is made up of many statements and clauses, you will need to be familiar with only a few to perform the examples in this chapter.

SELECT

The `SELECT` statement specifies the data you want to retrieve from a database. The `SELECT` statement uses the `FROM` clause to specify the name of the table that stores the data you want to retrieve. The `WHERE` clause specifies exactly which data you want to retrieve.

Example:

```
SELECT Total
FROM invoiceNumbers
WHERE Total > '$100'
```

INSERT

The `INSERT` statement allows you to add data to a database. The `INSERT` statement uses the `INTO` clause to specify the name of the table to which you want to add data and the names of the fields that store the data in the table. The `VALUES` clause specifies the values that you are adding.

Example:

```
INSERT INTO invoiceNumbers (INVOICE, TOTAL)
VALUES (12843, '$34.56')
```

DELETE

The `DELETE` statement is used to remove data from a database. The `DELETE` statement uses the `FROM` clause to specify the name of the table that stores the data you want to delete. The `WHERE` clause contains information that uniquely identifies the data you want to delete.

Example:

```
DELETE FROM invoiceNumbers
WHERE year < 1996
```

CREATE A DATA SOURCE NAME

If a Web server running a Windows operating system will be used to access a database you created, you must assign a Data Source Name (DSN) to the database.

A DSN stores information that tells Web applications how to access a specific database. You include the data source name in the JSP pages you want to connect to the database.

You only have to create a DSN once for a database. You do not have to create a new data source name when you change or update the structure of the database.

The data source name must be created on the Web server that will access both the database and the JSP pages that use the database. If a Web hosting service is storing your database and JSP pages, the Web hosting service will usually create the DSN for you.

To create a data source name, you specify the driver for the program you used to create the database, such as Microsoft Access or SQL Server. You then specify the DSN you want to use and the location of the database. The data source name does not have to be the same as the name of the database. You should use a short, descriptive DSN.

The steps below create a system DSN for a Microsoft Access database that will be accessed by a Web Server running the Windows 2000 operating system. Windows 2000 computers use a program labeled Data Sources to control DSN configuration. The name and location of the program used to create a DSN on your computer may be different, depending on the operating system you are using. For more information about how to create a DSN on your computer, refer to the computer's operating system documentation.

Extra

TYPES OF DATA SOURCE NAMES

There are three main types of data source names available on computers running a Windows operating system. The types of data source names differ in where the information about a database is stored and who can use the DSN. The administrator of the Web server usually specifies the type of DSN that must be used.

System DSN

The information in a system DSN is stored in the registry of the Web server. Any user that has access to the server will be able to use a system DSN to access the database.

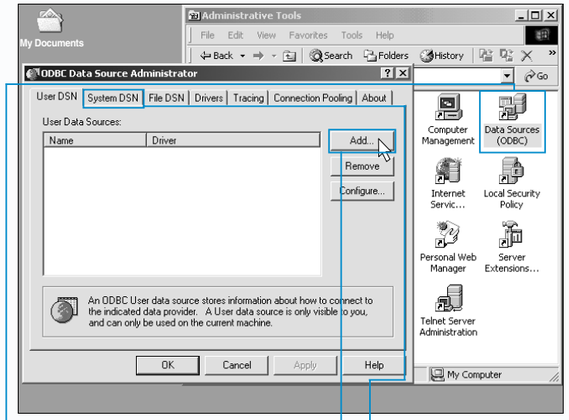
User DSN

The information in a user DSN is stored in the registry of the Web server, but only a specific user account can use the DSN. User data source names are often used when developing intranet Web applications that require secure access to a database.

File DSN

The information in a file DSN is stored in a text file on the Web server. File data source names make it easy to transfer databases and data source names between different Web servers. Any user who has access to the Web server will be able to use a file DSN to access the database.

CREATE A DATA SOURCE NAME



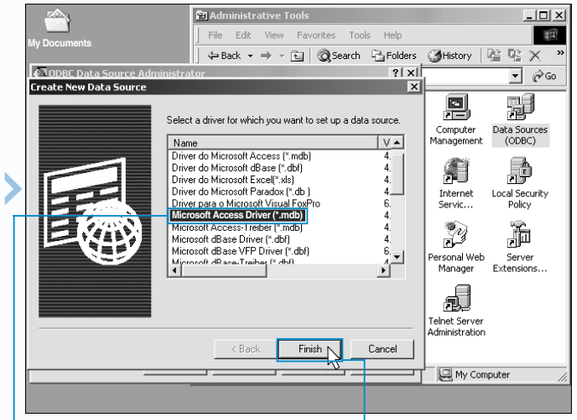
1 In the Control Panel, double-click Administrative Tools to display the Administrative Tools window.

2 Double-click Data Sources.

■ The ODBC Data Source Administrator dialog box appears.

3 Click the System DSN tab.

4 Click Add to create a data source name.

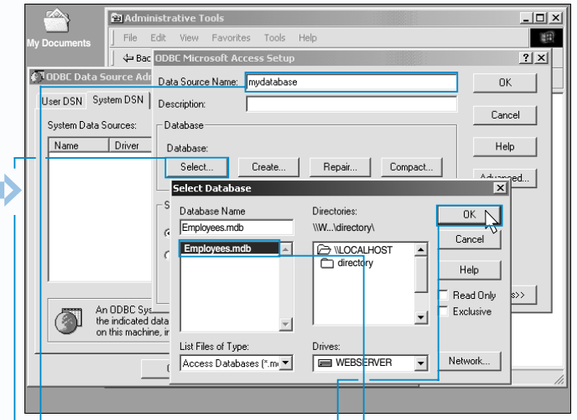


■ The Create New Data Source dialog box appears.

5 Click Microsoft Access Driver.

6 Click Finish.

■ The ODBC Microsoft Access Setup dialog box appears.

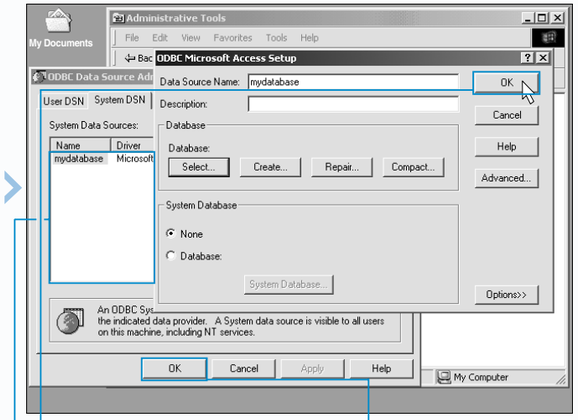


7 Type the data source name you want to use for the database.

8 Click Select to display the Select Database dialog box.

9 Select the database you want to create a data source name for.

10 Click OK in the Select Database dialog box.



11 Click OK in the ODBC Microsoft Access Setup dialog box.

■ The new data source name appears in this area.

12 Click OK to close the ODBC Data Source Administrator dialog box.

CONNECT TO A DATABASE

Once a Data Source Name (DSN) has been created for a database, you can set up a connection to the database in a JSP page. You can then use the JSP page to access the database. For example, the JSP page can be used to retrieve information from the database.

In order to set up a connection to a database, a driver that enables the JSP page to communicate with the database must be loaded. JavaServer Pages technology uses the Java DataBase Connectivity (JDBC) specification to access databases, while most databases created on computers using the Windows platform use the Open DataBase Connectivity (ODBC) specification. The Java SDK includes a JDBC-ODBC bridge driver that allows JSP pages to communicate with these Windows databases.

To load a driver in a JSP page, you use the `Class.forName` statement to specify the name of the driver. The name of the JDBC-ODBC bridge driver is `sun.jdbc.odbc.JdbcOdbcDriver`.

Once the driver has been loaded, a `Connection` object can be created that will allow the JSP page to connect to the database. Before a `Connection` object can be created, you must use the `page` directive to import the `java.sql` package. The `java.sql` package contains the `Connection` interface and is part of the Java class library. For more information about the `page` directive, see page 74.

The `DriverManager.getConnection` statement is used to specify the location of the database you want the JSP page to connect to. For connections created using the JDBC-ODBC bridge driver, the location will begin with `jdbc:odbc:` and be immediately followed by the DSN of the database. The `DriverManager.getConnection` statement also allows you to specify a login name and password if this information is required to establish a connection to the database.

The `close` method of the `Connection` object should be used to close a database connection when the connection is no longer needed.

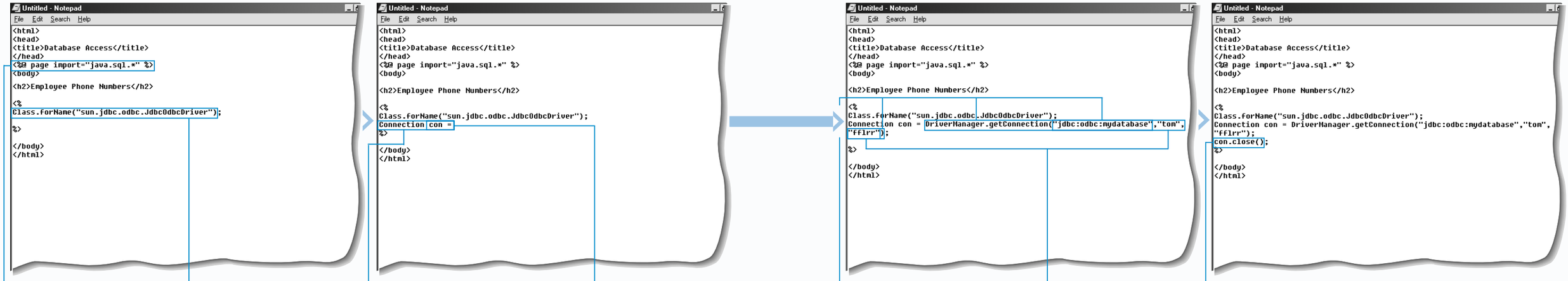
Extra

You must load a driver to connect a JSP page to a database even if the database uses the JDBC specification and does not require the use of the JDBC-ODBC bridge driver. Many database programs come with their own JDBC drivers. You may be able to load a database program's driver simply by specifying the name of the driver in the `Class.forName` statement. You should consult the documentation for the database program to determine which drivers are offered and how to load and use the drivers.

There is more than one version of the JDBC specification available. Version 2.0 is the latest version and includes features that are not found in older versions. You must ensure that your database is compatible with the JDBC version you intend to use. The Java SDK includes JDBC version 2.0.

Specifying a login name and password in a JSP page for a database connection can present a security risk, since anyone who has access to the JSP code will be able to determine this sensitive information. You may be able to use security features provided by your database program to minimize the security risk. For example, if the information in a database will only be retrieved, you may want to set up read-only access to the database. Consult the documentation for your database program for information on the available security features.

CONNECT TO A DATABASE



1 To import the `java.sql` package, type **<%@ page import="java.sql.*" %>**.

■ The `java.sql` package contains the `Connection` interface.

2 To load the bridge driver that allows the JSP page to communicate with a Windows database, type **Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver")**.

3 To create a `Connection` object that allows the JSP page to connect to a Windows database, type **Connection**.

4 Type a name for the `Connection` object followed by **=**.

5 To specify the location of the database you want the JSP page to connect to, type **DriverManager.getConnection()**.

6 Between the parentheses, type "**jdbc:odbc:** immediately followed by the DSN of the database. Then type ".

7 If the connection requires a login name and password, type a comma followed by the login name enclosed in quotation marks. Then type a comma followed by the password enclosed in quotation marks.

8 To close the conection to the database, type the name of the `Connection` object followed by a dot. Then type **close()**.

9 Save the page with the .jsp extension.

■ You can now use the JSP page to access a Windows database.

CREATE A RESULT SET

After setting up a connection to a database in a JSP page, you can create a result set to store information you retrieve from the database.

Before a JSP page can retrieve data from a database, the page must have permission to access the database. Permission to access a database from a JSP page is usually controlled by your operating system or database program. For information about access permissions, you should consult the documentation included with your software.

Before creating a result set, you must first create a `Statement` object that will retrieve information from a database. To create a `Statement` object, you use the `createStatement` method of the `Connection` object created when the database connection was set up. The `Statement` interface that is used to create the `Statement` object is part of the `java.sql` package.

Once the `Statement` object has been created, the results retrieved by the object must then be assigned to a `ResultSet` object. This object will be used to store the results returned from the database in a result set. To use the `ResultSet` object, you must create an instance of the object and assign it a name.

When the `ResultSet` object has been created, you can specify the information you want to place in the result set. To do this, you use the `executeQuery` method of the `Statement` object to issue a `SELECT` statement to the database. The `SELECT` statement allows you to specify the data you want to retrieve from a table in the database. You can specify the data you want to retrieve by name or use an asterisk (`*`) to retrieve all the data in the table. The `SELECT` statement uses the `FROM` clause to specify the name of the table that stores the information you want to retrieve.

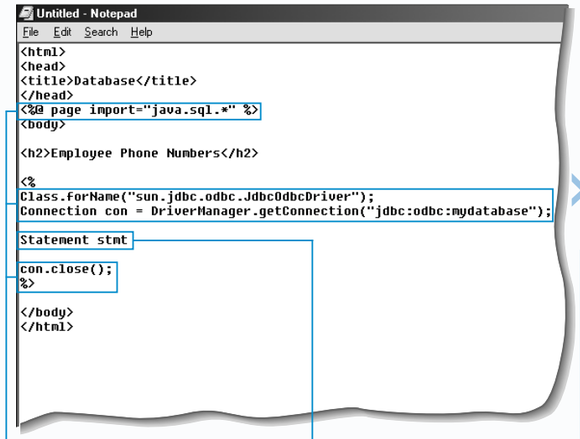
Extra

You can use any name you want for your database objects. However, there are some names that are usually used for certain common objects. For example, the `Connection` object is often named con and the name stmt is often used for the `Statement` object. The `ResultSet` object is usually named rs.

Depending on the size, speed and location of the database, it may take a long time for a JSP page to pass a `SELECT` statement to the database, process the statement and then retrieve the results generated from the database. You should take this time into account when designing your JSP pages. For example, if your JSP page displays a banner image followed by a large amount of data from a database, you can use the `flush` method of the `out` object to force the JSP page to display the banner first, while the database information is being retrieved.

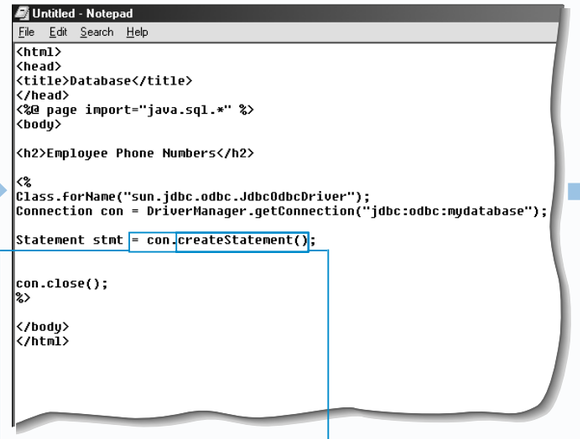
In order to minimize delays when communicating with a database, you should design your SQL statements to be efficient. For example, if you require data only from a particular field in a database, the `SELECT` statement should retrieve only the relevant information. It is much more efficient to retrieve only the data you need from the database than to retrieve unnecessary information and then filter the results.

CREATE A RESULT SET



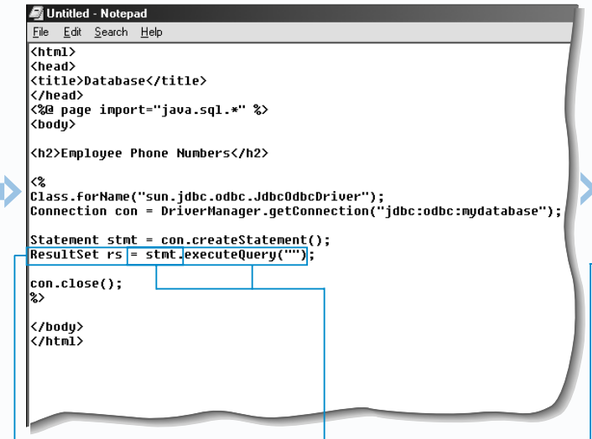
1 Type the code that creates a connection to the database from which you want to retrieve information.

2 To create the `Statement` object that will retrieve information from the database, type **Statement** followed by a name for the `Statement` object.



3 Type = and the name for the `Connection` object followed by a dot.

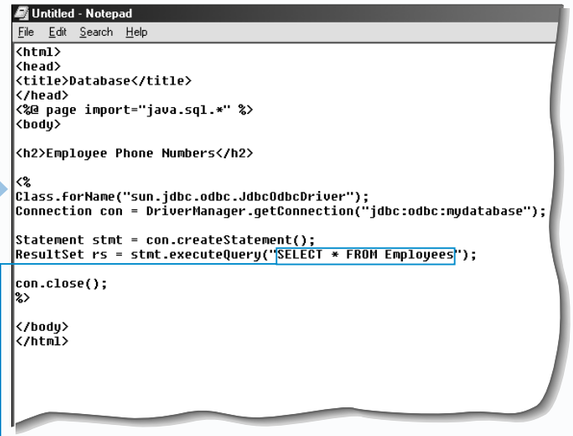
4 Type **createStatement**().



5 To create a `ResultSet` object to store the results returned from the database, type **ResultSet** followed by a name for the `ResultSet` object.

6 Type = and the name for the `Statement` object followed by dot.

7 Type **executeQuery**("").



8 Between the quotation marks, type **SELECT * FROM** followed by the name of the table in the database from which you want to retrieve information.

9 Save the page with the .jsp extension.

■ To retrieve the data from the result set, see page 150.

RETRIEVE DATA FROM A RESULT SET

Once information has been retrieved from a database and placed in a result set, you can retrieve the data from the result set. A result set consists of rows which store information generated by the database when an SQL statement is processed.

Information may be accessed in the result set one row at a time. An imaginary indicator, called a cursor, is used to identify which row can currently be accessed. When the rows of data are initially placed in a result set, the cursor is placed just above the first row of data. To access the first row of data in a result set, you must call the `next` method of the `ResultSet` object to move the cursor to the first row.

If a result set contains multiple rows, a loop is typically used to retrieve information from each row. The `next` method of the `ResultSet` object is usually used in conjunction with a `while` loop to move the cursor through the rows of data in the result set, one at a time. The `next` method returns a boolean value which indicates if another row to which the cursor can be moved to exists. If the `next` method returns a `true` value, the loop continues and the next row of data is processed.

When the cursor is positioned in a particular row of data, a method of the `ResultSet` object can be used to retrieve information from that row. For example, the `getString` method can be used to retrieve string information from a row of data. When using the `getString` method, you must specify the name of the field from which you want to retrieve data. You can assign the value returned by the `getString` method to a variable, which allows you to use the value in a process or to display the value in a Web browser.

Extra

In addition to string data, a result set can also contain other types of data such as objects and primitive data types. Different methods of the `ResultSet` object are used to access different data types.

Example:

```
int numberOfItems = rs.getInt("quantity");
double itemPrice = rs.getDouble("price");
```

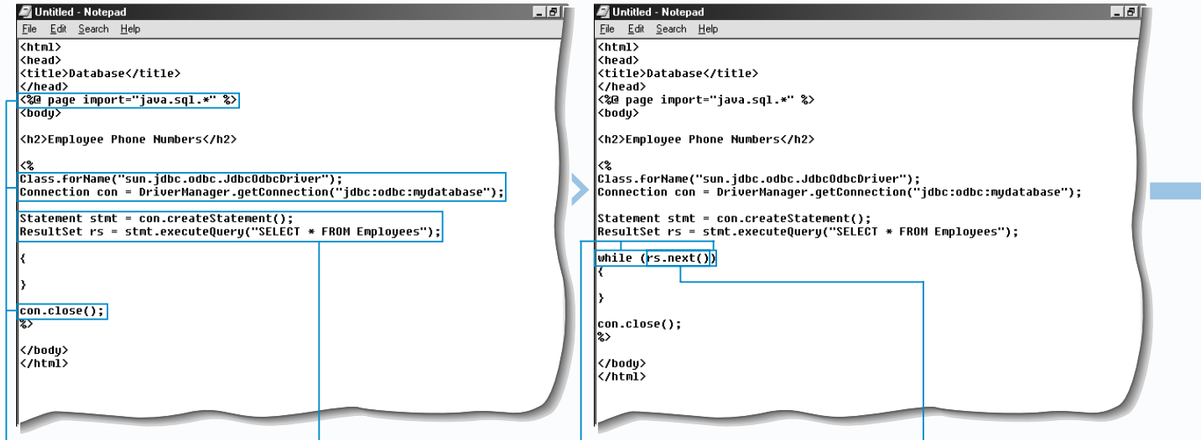
If multiple columns in the same result set have the same name, the method used to retrieve the data from the result set will retrieve the data from the first column that has the common name. Although it is not recommended, it is possible to have multiple columns with the same name in a database.

You can also use a column number instead of a field name to retrieve information from a row of data. In a result set, the first column of information has a column number of 1, not 0 as some programmers might expect.

Example:

```
while (rs.next())
{
      String employeeId = rs.getString(1);
      String employeeName = rs.getString(2);
      String employeeExt = rs.getString(3);
}
```

RETRIEVE DATA FROM A RESULT SET



1 Type the code that creates a connection to the database from which you want to retrieve information.

2 Type the code to create a `Statement` object that retrieves information from a database and to create a result set that stores the results returned from the database.

3 To create a `while` loop to cycle through the rows of data in the result set, type **while** ().

4 Between the parentheses, type the name of the `ResultSet` object followed by **.next**().

5 In the body of the `while` loop, type **String** followed by the name of a variable you want to assign the string data to.

6 Type **=** and the name of the `ResultSet` object followed by a dot.

7 To retrieve string information from the row, type **getString**("").

8 Between the quotation marks, type the name of the field that holds the information you want to retrieve.

9 Repeat steps 5 to 8 to retrieve the information you want from the result set.

10 Save the page with the .jsp extension.

■ To format the retrieved data for display in a Web browser, see page 152.

FORMAT DATA FOR DISPLAY

Once information has been retrieved from a database and accessed from the result set, the information can be formatted for display on a JSP page. When displaying information retrieved from a database, you can use HTML tags to format the information. For example, HTML tags can be used to place the information in a list or table.

If a result set contains multiple rows, a loop is typically used to retrieve information from each row, one at a time. The `next` method of the `ResultSet` object is usually used in conjunction with a `while` loop to move the cursor through the rows of data.

When the cursor is positioned in a particular row of data, a method of the `ResultSet` object can be used to retrieve information from that row. For example, the `getString` method can be used to retrieve string information from a field you specify in the current row.

Assigning the value of the `getString` method to a variable can make it easier to work with the data. You can use the `print` method of the `out` object to display the contents of the variable on a JSP page. When using the `print` method, different types of data, such as variables and string literals, can be joined together using the concatenation operator, `+`.

You can incorporate any HMTL code you want to use into the loop that accesses each row of data so that with each iteration of the loop, a row of data and the HTML code used to format the data will be sent to the client.

Extra

Many Web pages on the Internet are not static pages, but rather are made up of information retrieved from databases. This information is assembled on a page each time a client views the page. For example, the home page of a news organization may contain information retrieved from a news database, a weather database and an advertising database. The information from each database is formatted with HTML tags and the separate sections are all joined together to create a single, seamless page.

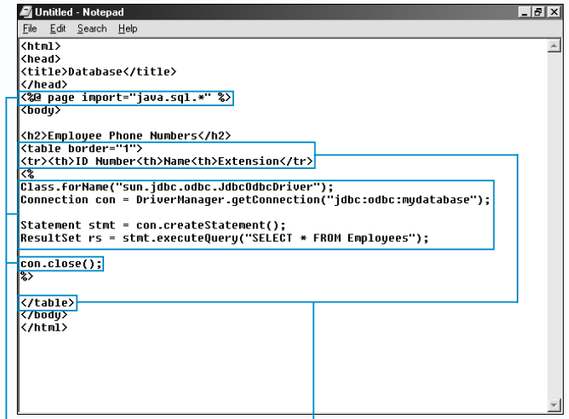
When formatting information retrieved from a database for display on a JSP page, you should first sketch out the desired layout of the page to ensure proper placement of information. If the amount of information retrieved from the database will vary with each query, you must take this into account when laying out the page.

If the information you want to display from a database is relatively simple, you can use an expression to display the information directly from the result set, without first assigning the information to variables.

Example:

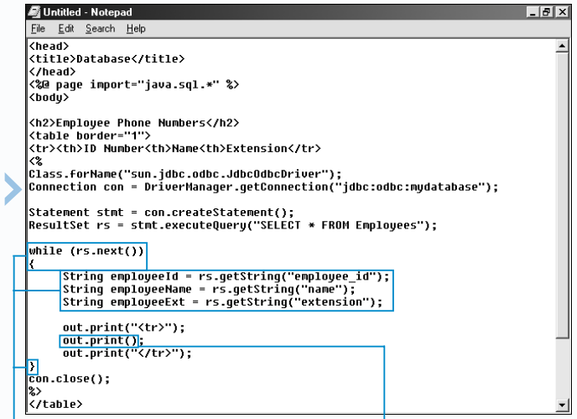
```
<table>
<tr>
<td><%= rs.getString("employee_id") %></td>
<td><%= rs.getString("name") %></td>
<td><%= rs.getString("extension") %></td>
</tr>
</table>
```

FORMAT DATA FOR DISPLAY



1 Type the code that creates a connection to a database and retrieves information from the database.

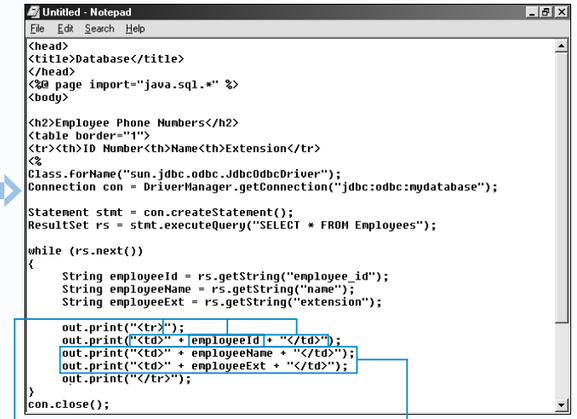
2 To display the information you retrieve from the result set in a table, type the HTML code that sets up the table.



3 Type the code that creates a loop that will process one row of the result set at a time.

4 Type the code that retrieves the data you want to display from the result set.

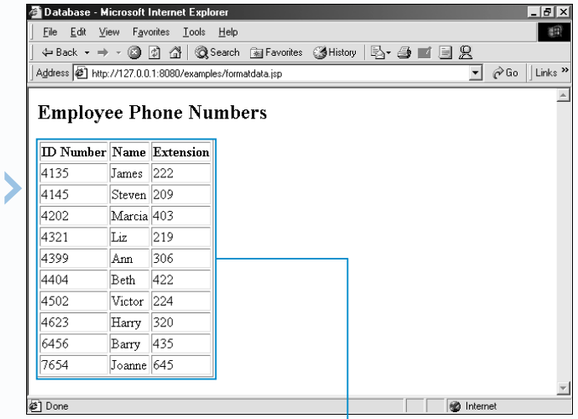
5 To display an item of information, type **out.print()**.



6 Between the parentheses, type the name of the variable that stores the information you want to display.

7 To generate a cell in the table, type the required HTML tags, enclosed in quotation marks. Separate each tag and variable with the concatenation operator.

8 Repeat steps 5 to 7 for each item of information you want to display.



9 Save the page with the .jsp extension and then display the page in a Web browser.

■ The Web browser displays the formatted information retrieved from a database.

POSITION THE CURSOR IN A RESULT SET

The `ResultSet` object provides several methods that can be used to move the cursor to a particular row in a result set.

Initially, the cursor is positioned above the first row in a result set, so there is no current row. You must call a method of the `ResultSet` object to move the cursor to the row you want to make current. The values in the current row are affected by any methods that are called.

If the `next` method of the `ResultSet` object is used to move the cursor forward through each row in a result set, a new result set would have to be created to revisit a row or iterate through the entire result set a second time. Most new JDBC drivers allow you to create a scrollable result set. You can move the cursor forward, backward and to a specific row in a scrollable result set.

To make a result set scrollable, you must specify the result set type as `TYPE_SCROLL_INSENSITIVE` or `TYPE_SCROLL_SENSITIVE`. If you want to be able to change information in the result set, you must also specify the concurrency type as `CONCUR_UPDATABLE`. The result set type and concurrency type are specified as arguments of the `createStatement` method. The values available for both of these types are constants determined by the `ResultSet` interface.

After setting the result set type and concurrency type, you can call a `ResultSet` method to position the cursor at the row you want to make the current row. Calling the `first` method moves the cursor to the first row in the result set. Calling the `last` method moves the cursor to the last row. To position the cursor at a specific row, you use the `absolute` method to specify the number of the row you want to make current.

Extra

You can display the result set type and concurrency type of a result set in a JSP page. To do so, use the `getType` and `getConcurrency` methods of the `ResultSet` object in the JSP page, such as `<%= rs.getType() %>` and `<%= rs.getConcurrency() %>`. When the JSP page is displayed in a Web browser, a numerical value appears, representing the result set type and concurrency type.

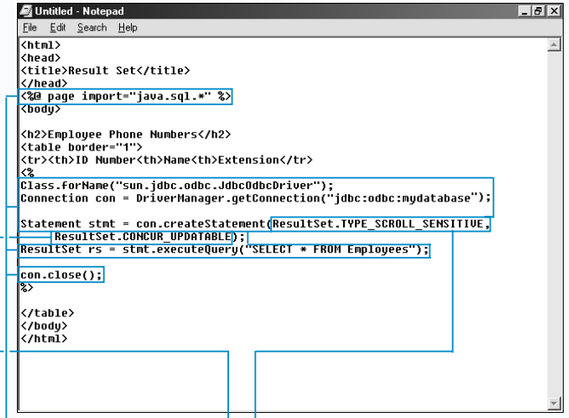
Result Set Types

VALUE:		DESCRIPTION:
1003	TYPE_FORWARD_ONLY	The result set is not scrollable. The cursor can move forward from top to bottom only.
1004	TYPE_SCROLL_INSENSITIVE	The result set is scrollable. Any changes made to the database while the result set is open are not reflected in the result set.
1005	TYPE_SCROLL_SENSITIVE	The result set is scrollable. Any changes made to the database are immediately reflected in the result set.

Concurrency Types

VALUE:		DESCRIPTION:
1007	CONCUR_READ_ONLY	The information in the result set cannot be modified.
1008	CONCUR_UPDATABLE	The information in the result set can be updated.

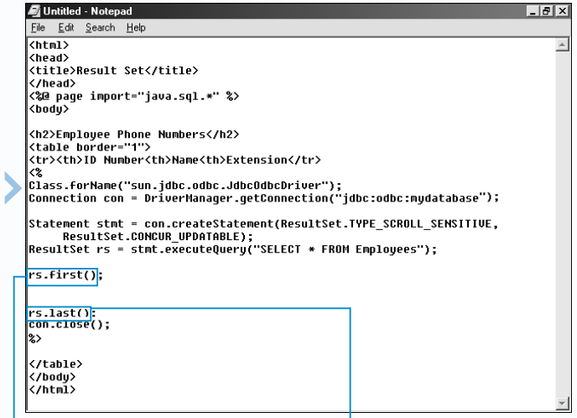
POSITION THE CURSOR IN A RESULT SET



1 Type the code that creates a connection to a database and retrieves information from the database.

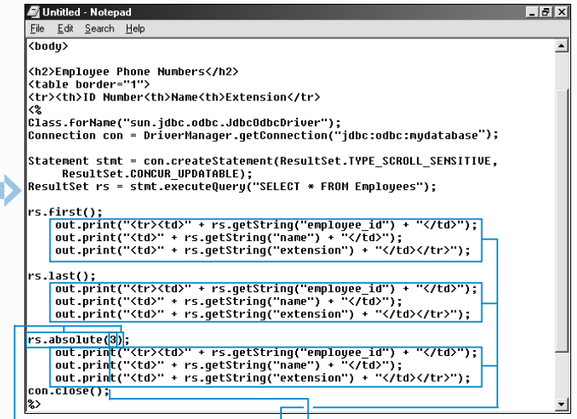
2 Between the parentheses for the `createStatement` method of the `Connection` object, type **ResultSet.TYPE_SCROLL_SENSITIVE** followed by a comma to specify the result set type.

3 Type **ResultSet.CONCUR_UPDATABLE** to specify the concurrency type.



4 To create a method that moves the cursor to the first row in the result set, type the name of the `ResultSet` object followed by a dot. Then type **first()**.

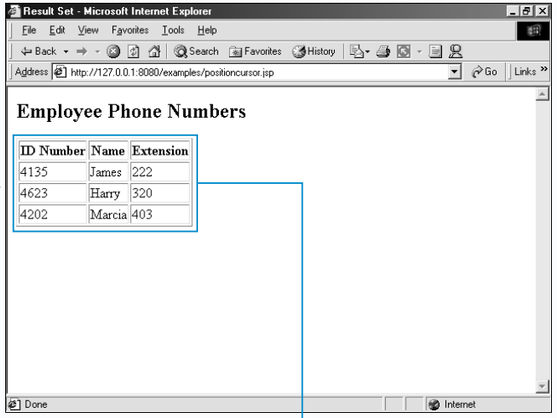
5 To create a method that moves the cursor to the last row in the result set, type the name of the `ResultSet` object followed by a dot. Then type **last()**.



6 To create a method that moves the cursor to a specific row in the result set, type the name of the `ResultSet` object followed by a dot. Then type **absolute()**.

7 Between the parentheses, type the number of the row you want to move the cursor to.

8 Type the code that retrieves and displays information from each row you specified in the result set.



9 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of positioning the cursor in a result set. The first, last and third rows in the result set are displayed.

ADD A RECORD

The `ResultSet` object provides methods you can use to insert records into a table in a database.

You insert a record into a table by inserting a new row into the result set that contains information retrieved from the database. The result set must contain all the columns in the table that are to be given values for a record. A column that is not included in the result set will be given a `null` value when the record is inserted into the table. If the column does not accept `null` values, an error will occur.

Before you can add a record, you must first use the `moveToInsertRow` method of the `ResultSet` object to position the cursor at the insert row. The insert row allows you to create a new row in a result set.

Once the cursor is positioned at the insert row, you can specify the values you want to add to each column in the row using special update methods of the `ResultSet` object. The method name you use depends on the type of data to be used for the value. For example, if you want to specify a string value for a column, you use the `updateString` method. To specify an integer value, you use the `updateInt` method.

Each update method requires two arguments. The first argument specifies the name or number of the column you want to contain the data. The number of the first column in the table is 1. The second argument specifies the value that will be inserted into the column. The data type of the value must match the update method you specified.

Once the values have been specified for each column in the table, you can call the `insertRow` method of the `ResultSet` object to add the new record to the result set and to the table in the database.

Extra

In order to add a record to a table in a database, the database driver must support the `insertRow` method. If errors occur when calling the `insertRow` method, you should check whether a version of the database driver that supports the method is available for your database program.

If you do not want to use the update methods of the `ResultSet` object to add a record, you can use the SQL `INSERT` command instead. You issue the `INSERT` command to a database using the `executeUpdate` method of the `Statement` object.

Example:

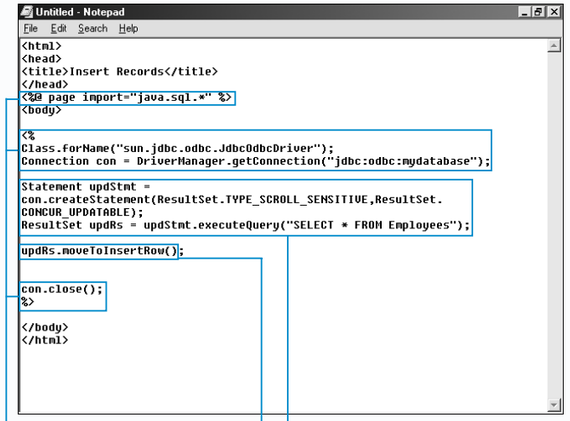
```
stmt.executeUpdate("INSERT INTO employees VALUES(4347, 'Peter', 456)");
```

You can access other rows in a result set to which you are adding a new row. When you finish inserting a row, you can move the cursor to any row in the result set. For example, you can use the `moveToCurrentRow` method of the `ResultSet` object to reposition the cursor at the last row accessed before you inserted the new record. To avoid losing the information you added to the insert row, you should move the cursor only after calling the `insertRow` method.

Example:

```
updRs.insertRow();
updRs.moveToCurrentRow();
```

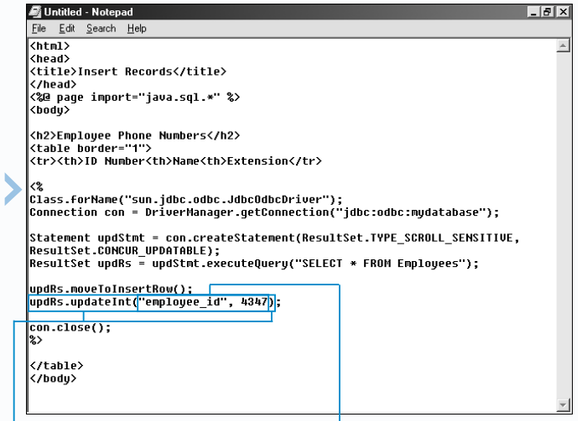
ADD A RECORD



1 Type the code that creates a connection to a database to which you want to add a record.

2 Type the code that retrieves information from the table where you want to add a record and allows you to update the database.

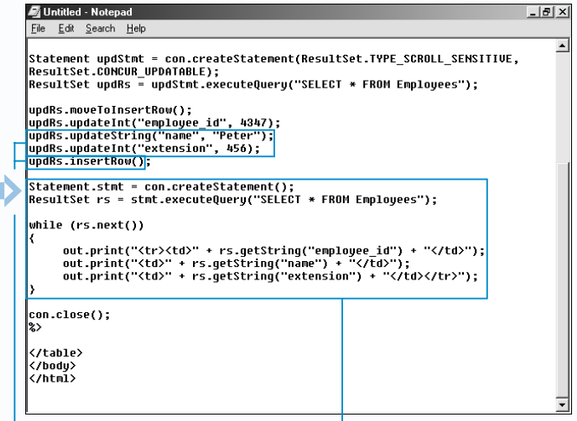
3 To position the cursor at the insert row, type the name of the `ResultSet` object followed by **.moveToInsertRow()**.



4 To specify a value for a column in the new record, type the name of the `ResultSet` object followed by a dot. Then type the update method you want to use followed by ().

5 Between the parentheses, type the name or number of the column to which you want to add data followed by a comma. Then type the value you want the column to contain.

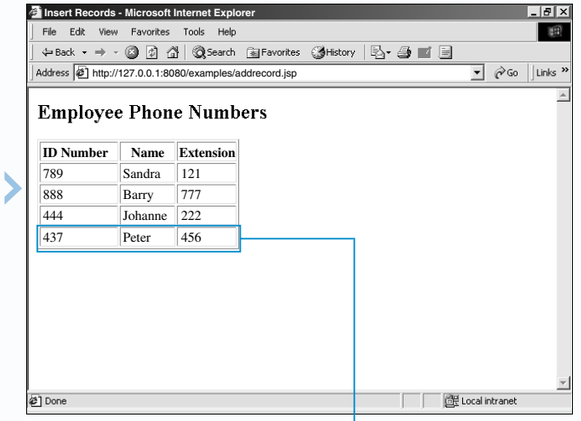
■ String arguments must be enclosed in quotation marks.



6 Repeat steps 4 and 5 for each value you want to insert for the record.

7 To insert the record into the result set and the database, type the name of the `ResultSet` object followed by **.insertRow()**.

8 Type the code that displays the information from the database.



9 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of adding a record to a table in a database.

ADD FORM DATA TO A DATABASE

A JSP page that contains a connection to a database can be used to add records to the database. Records are commonly added using data submitted by forms. Forms provide an easy-to-use interface for working with a database.

The `getParameter` method of the `request` object can be used in a JSP page to access data passed by a form. For more information about the `getParameter` method, see page 84.

When creating a result set to add a record to a database, you must set the result set type and concurrency type. For information about setting the result set type and concurrency type, see page 154.

The SQL `INSERT` statement allows you to add a record to a database. The `INSERT` statement uses the `INTO` clause to specify the name of the database table you want to add a record to and the names of the fields that store information in the table. The `VALUES` clause specifies the field values that make up the record you are adding to the database. You may have to enclose the field values in single or double quotation marks, depending on your database program.

It is common programming practice to store an SQL `INSERT` statement in a variable. Using variables can help make your code easier to read and update.

The SQL `INSERT` statement is executed by the `executeUpdate` method of the `Statement` object to add data to the database.

When creating the code for a form that will be used to add records to a database, you must specify the name of the JSP page that connects to the database in the `action` attribute of the `<form>` tag.

Apply It

Using an `if` statement allows you to confirm that information has been submitted by a form before the JSP page connects to the database that stores the form data. For example, you can ensure that a user name entered into a form contains at least one character before the JSP page sends any information to the database.

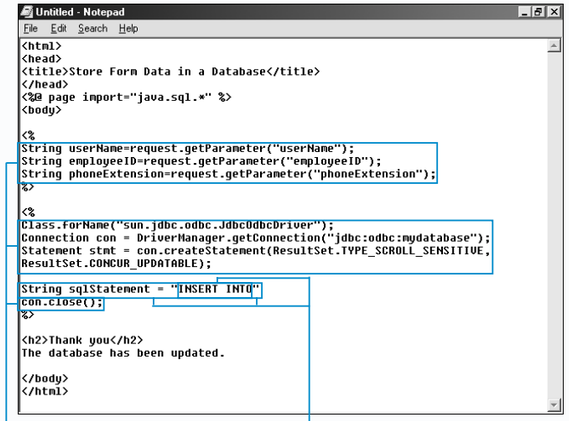
TYPE THIS:

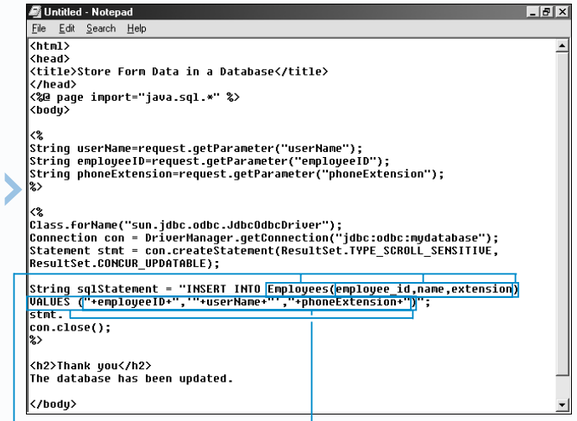
```
if (userName.length()>0)
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:mydatabase");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

String sqlStatement = "INSERT INTO Employees(employee_id,name,extension) VALUES
("+employeeID+",'"+userName+"','"+phoneExtension+")";
stmt.executeUpdate(sqlStatement);
}
else
{
out.print("Please enter a user name.");
}
```

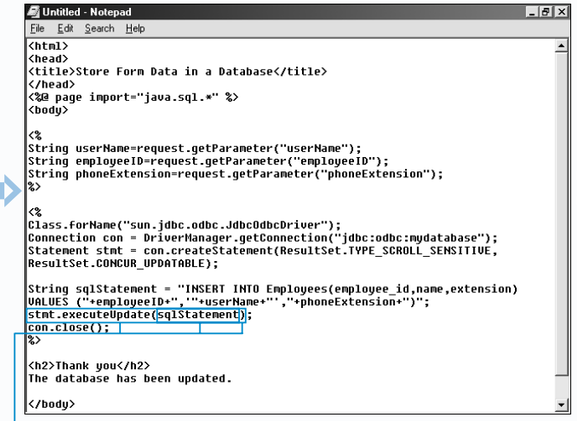
RESULT:

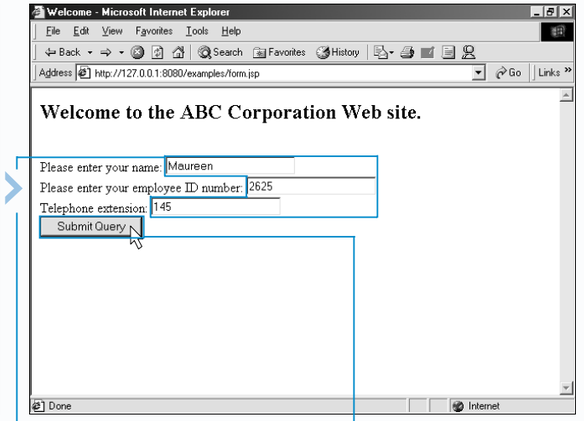
Please enter a user name.

ADD FORM DATA TO A DATABASE









1 Type the code that accesses information passed to the JSP page by a form.

2 Type the code that creates a connection to the database you want to add records to and creates a result set.

3 Type the code that creates a variable to store the SQL `INSERT` statement followed by **=""**.

4 Between the quotation marks, type **INSERT INTO**.

5 Type the name of the table in the database that you want to add records to followed by **()**.

6 Between the parentheses, type the name of each field in the table, separated by a comma.

7 Type **VALUES()**.

8 Between the parentheses, type the code that uses the information passed by the form.

9 Type the name of the `Statement` object followed by a dot. Then type **executeUpdate()**.

10 Between the parentheses, type the variable name that stores the SQL `INSERT` statement.

11 Save the page with the .jsp extension.

12 In a Web browser, display the form you created to add records to the database.

13 Enter data into the form.

14 Click the submit button to pass the data in the form to the JSP page.

■ The JSP page that adds the record will appear and the record will be added to the database.

UPDATE A RECORD

Once you establish a connection with a database, you can edit the information contained in the database. Editing the information in a database allows you to keep the information up-to-date.

If you want to update a single record, you can use the `WHERE` clause with the SQL `SELECT` command to create a result set that stores only the row of data you want to update.

When a result set is created, the cursor is positioned above the first row of data. Before information in the result set can be modified, you must use the `next` method of the `ResultSet` object to move the cursor to the row that is to be updated, even if the result set contains only a single row.

You can specify the values you want to change for the current row using special update methods of the `ResultSet` object. The method name you use depends on the type of data to be used for the value. For example, if you want to specify a string value, you use the `updateString` method. To specify an integer value, you use the `updateInt` method.

Each update method requires two arguments. The first argument specifies the name of the column you want to contain the data. The second argument specifies the value that will be inserted into the column. The data type of the value must match the update method you specified.

Once the update methods have been used to specify the data you want to change in the current record, you can call the `updateRow` method of the `ResultSet` object to update the information in the database.

Extra

If you do not want to use the update methods of the `ResultSet` object to update a record, you can use the SQL `UPDATE` statement instead. You issue the `UPDATE` command to a database using the `executeUpdate` method of the `Statement` object.

Example:

```
stmt.executeUpdate("UPDATE Employees SET
name = 'Pete' WHERE (name = 'Peter')");
```

You can cancel updates to a database by using the `cancelRowUpdates` method. The `cancelRowUpdates` method can be called after any update methods are used, but before the `updateRow` method is called. Canceling updates is useful if the JSP code detects invalid data or a database access error.

Example:

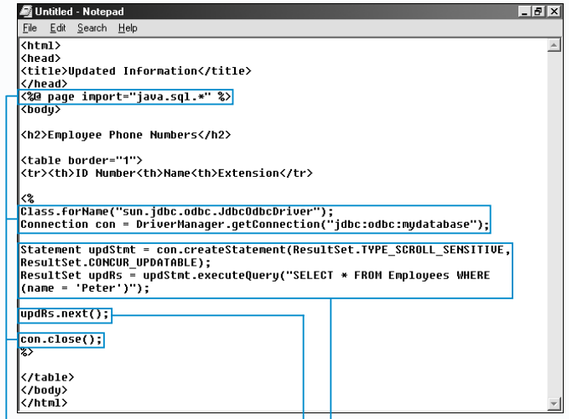
```
updRs.updateString("name", "Pete");
updRs.cancelRowUpdates();
```

You can also use column numbers instead of column names to specify the columns you want to update in a record. In SQL, column numbers start at column 1, not 0 like many other indexing systems used in programming.

Example:

```
updRs.next();
updRs.updateString(2, "Pete");
updRs.updateInt(3, 456);
updRs.updateRow();
```

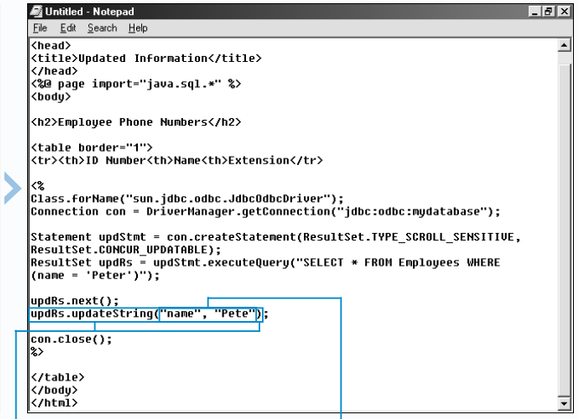
UPDATE A RECORD



1 Type the code that creates a connection to a database in which you want to update records.

2 Type the code that retrieves a record from the table you want to update and allows you to update the database.

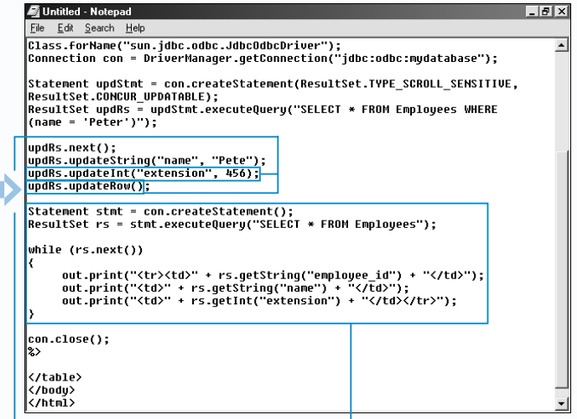
3 Type the code that moves the cursor to the row you want to update.



4 To specify a new value for a column in the record you want to update, type the name of the `ResultSet` object followed by a dot. Then type the update method you want to use followed by ().

5 Between the parentheses, type the name of the column you want to update followed by a comma. Then type the new value you want the column to contain.

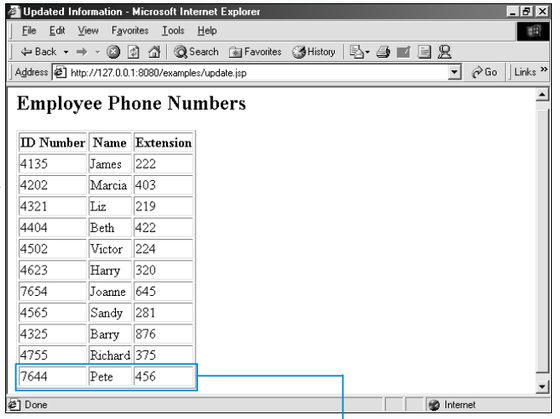
■ String arguments must be enclosed in quotation marks.



6 Repeat steps 4 and 5 for each column you want to update.

7 To change the information in the database, type the name of the `ResultSet` object followed by **.updateRow()**.

8 Type the code that displays the information in the database.



9 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of updating a row in a database.

MAKE A BATCH UPDATE

SQL statements are usually sent to a database program one at a time and the program processes each SQL statement as it is received. In most cases, this is an acceptable way of processing SQL statements. However, for some larger databases, it may be more efficient to combine individual SQL statements together in a batch that is sent at one time. This is especially useful when sending multiple update statements to a database program.

When a connection to a database is established, the connection is usually configured to send each SQL statement to the database program as it is created. To make batch updates, you must set up the `Connection` object so the connection will wait for a specific instruction before sending the SQL statements to the database program. To do this, you must set the parameter of the `setAutoCommit` method of the `Connection` object to `false`.

An SQL statement is added to a batch using the `addBatch` method of the `Statement` object. The argument of the `addBatch` method must be a valid SQL statement, although the statement cannot return a result set. Once all the SQL statements you want to send to the database program have been added to the batch, the batch can be sent to the database program using the `executeBatch` method of the `Connection` object.

After the `executeBatch` method is called, you must also call the `commit` method of the `Connection` object to make any changes to the database permanent. If the `commit` method is not called, the changes made by the batch update will still be reflected in the result set, but the changes will not be permanently written to the database.

Extra

You should make sure that the database driver used to communicate with the database program is able to perform batch operations before using the `executeBatch` method. If the database driver does not support batch operations, you should check if a newer version of the driver is available.

After using the `commit` method of the `Connection` object to make your changes to a database permanent, you may need to re-enable the auto-commit mode of the `Connection` object. To do so, you must set the parameter of the `setAutoCommit` method of the `Connection` object to `true`.

Example:

```
updStmt.executeBatch();
con.commit();
con.setAutoCommit(true);
```

The `executeBatch` method returns an array of integers that indicates the number of records affected by each SQL statement in a batch. The value of the first element in the array corresponds to the first SQL statement in the batch, and so forth.

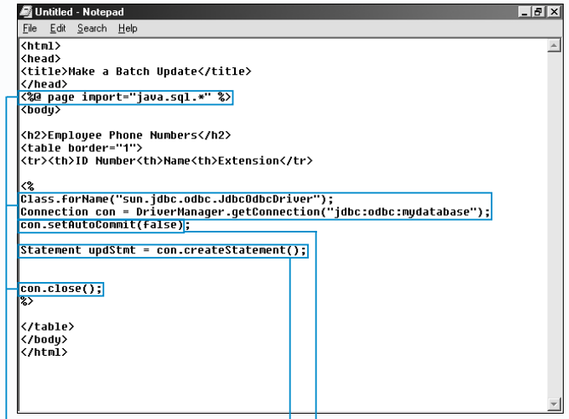
TYPE THIS:

```
updStmt.addBatch("DELETE FROM Employees WHERE name='Martine'");
updStmt.addBatch("DELETE FROM Employees WHERE name='Tom'");
int[] returnValues = updStmt.executeBatch();
for (int x = 0; x < returnValues.length; x++)
{
      out.print("SQL statement #" + (x+1) + " deleted ");
      out.print(returnValues[x] + " record(s)");
}
```

RESULT:

```
SQL statement #1 deleted 1 record(s)
SQL statement #2 deleted 1 record(s)
```

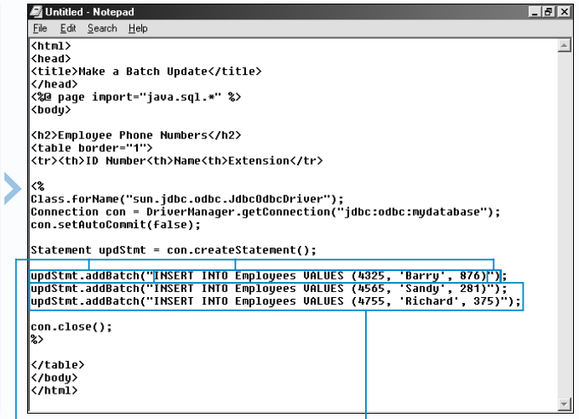
MAKE A BATCH UPDATE



1 Type the code that creates a connection to the database to which you want to make a batch update.

2 To disable the auto-commit mode of the `Connection` object, type **con.setAutoCommit(false)**.

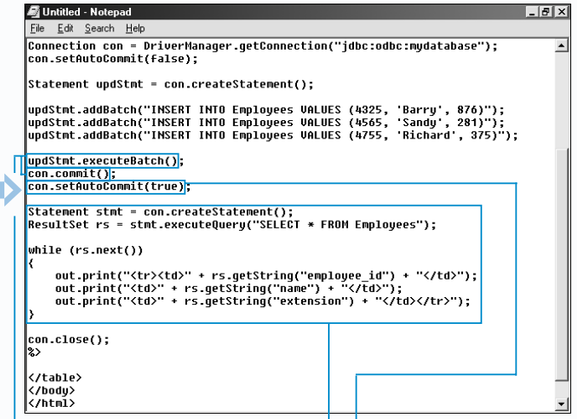
3 Type the code that creates a `Statement` object for the batch.



4 To add an SQL statement to the batch, type the name of the `Statement` object followed by **.addBatch("")**.

5 Between the quotation marks, type a valid SQL statement.

6 Repeat steps 4 and 5 for each SQL statement you want to add to the batch.

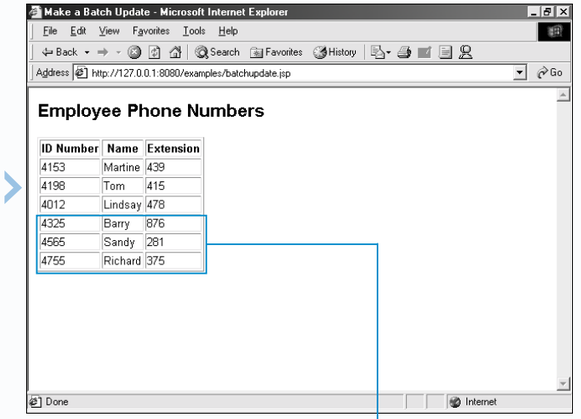


7 To send the batch update to the database program, type the name of the `Statement` object followed by **.executeBatch()**.

8 To make any changes to the database permanent, type the name of the `Connection` object followed by **.commit()**.

9 To re-enable the auto-commit mode of the `Connection` object, type **con.setAutoCommit(true)**.

10 Type the code that retrieves information from the updated database and displays the information on the JSP page.



11 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of making a batch update.

CREATE A PREPARED STATEMENT

Before a database program can execute an SQL statement sent from a JSP page, the SQL statement is compiled into a form that is understood by the inner workings of the database program. Compiling an SQL statement can be a relatively lengthy process, but you can create a prepared statement to save time in accessing the database after the initial query is processed.

A prepared statement is an SQL statement that is precompiled by a database program. A prepared statement needs to be compiled only once, so it is very useful in cases where the same SQL statement will be sent to the database program numerous times.

You use a `PreparedStatement` object to send an SQL statement that you want to precompile to a database program. A `PreparedStatement` object is created using the `prepareStatement` method of the `Connection` object. The `prepareStatement` method takes the SQL statement you want to precompile as an argument.

You use the `executeQuery` method of the `PreparedStatement` object to instruct the database program to process the SQL statement that has been precompiled. The result generated when a prepared statement is processed is usually assigned to a `ResultSet` object.

Depending on the SQL statement you are precompiling, you may need to specify the result set type and concurrency type of the result set in the `prepareStatement` method. For information about specifying result set and concurrency types, see page 154.

If the SQL statement you want to precompile requires parameters, you must set up the prepared statement to accept parameters. For information about using parameters in a prepared statement, see page 166.

Extra

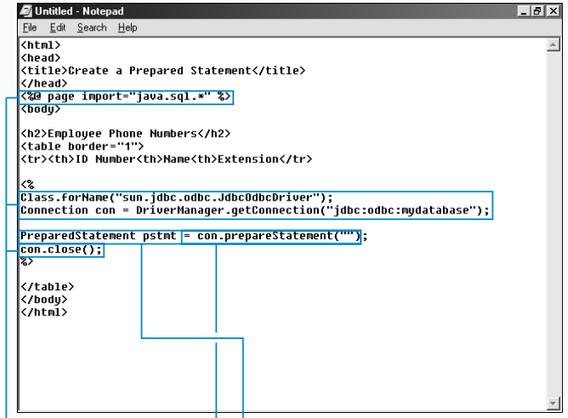
When using the `executeQuery` method, a result set is generated to store the results of the query. The `execute` method of the `PreparedStatement` object can be used instead of the `executeQuery` method to execute a prepared statement that does not return a result. For example, you may use the `execute` method for an SQL statement that removes a table from a database.

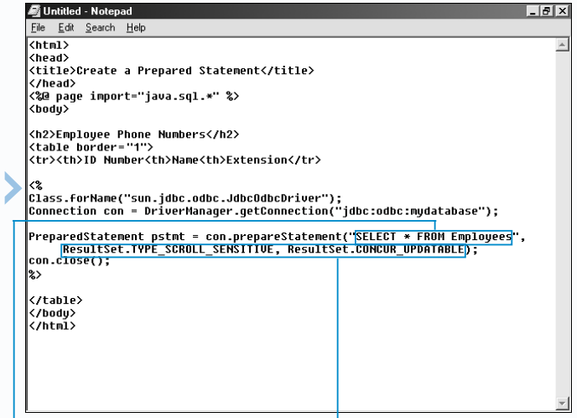
Example:

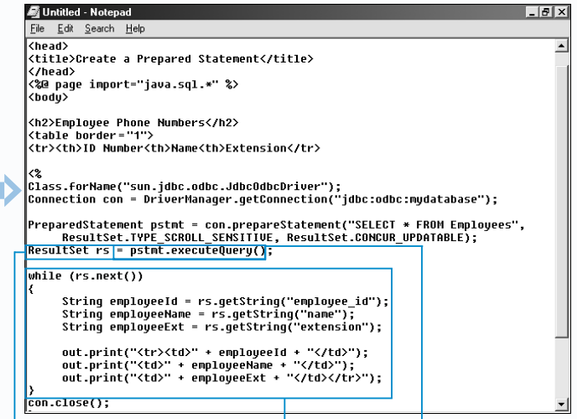
```
PreparedStatement pstmt = con.prepareStatement("DROP TABLE Employee");
pstmt.execute();
```

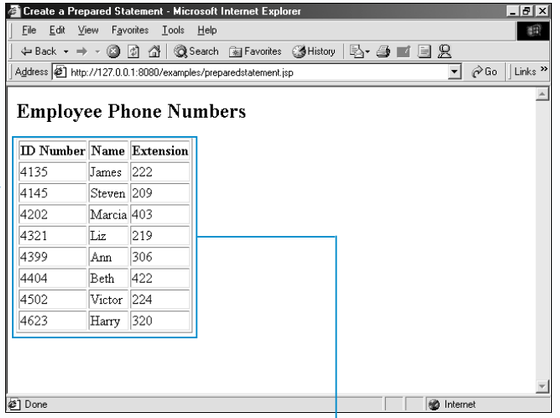
Like the `Statement` object, the `PreparedStatement` object can use the `execute`, `executeQuery` and `executeUpdate` methods to execute SQL statements. These methods do not require any arguments when used with a `PreparedStatement` object because the SQL statement is specified when the `PreparedStatement` object is created. When the same methods are used with a `Statement` object however, an SQL statement is usually passed to the methods as an argument. For more information about the methods supported by the `Statement` and `PreparedStatement` objects, refer to the `java.sql` package documentation.

CREATE A PREPARED STATEMENT









1 Type the code that creates a connection to the database to which you want to send a prepared statement.

2 To create a `PreparedStatement` object, type **PreparedStatement** followed by a name for the `PreparedStatement` object.

3 Type **=** and the name of the `Connection` object followed by **.prepareStatement("")**.

4 Between the quotation marks, type the SQL statement to be precompiled.

5 If necessary, type the code that specifies the result set type and concurrency type for the `ResultSet` object that will store the results returned from the database.

6 To create a `ResultSet` object to store the results returned from the database, type **ResultSet** followed by a name for the `ResultSet` object.

7 Type **=** and the name of the `PreparedStatement` object followed by **.executeQuery()**.

8 Type the code that retrieves the data from each row of the result set and formats the data for display.

9 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of using a prepared statement to send SQL statements to a database.

USING PARAMETERS IN A PREPARED STATEMENT

Prepared statements are ideal for repeatedly sending the same SQL statements to a database. Typically, prepared statements are used with SQL statements that have parameters. For example, an SQL statement can be used to add records to a database. The structure of the SQL statement remains the same for each record that is added, but the values of the parameters change each time the statement is executed.

The `PreparedStatement` object is used to issue an SQL statement that contains one or more parameters to a database. When creating a prepared statement that uses parameters, you use question marks to indicate where you want to place parameter values in the SQL statement. There is no limit to the number of question marks you can use in an SQL statement.

Before the SQL statement can be executed, the values for the question marks must be specified using special set methods of the `PreparedStatement` object. The method name you use depends on the type of data to be used for the value. For example, if you want to specify a string value for a question mark, you use the `setString` method. To specify an integer value, you use the `setInt` method.

Each set method requires two arguments. The first argument specifies the position of the question mark in the SQL statement. The position of the first question mark in an SQL statement is 1. The second argument specifies the value that will replace the question mark in the SQL statement. The data type of the value must match the set method you specified.

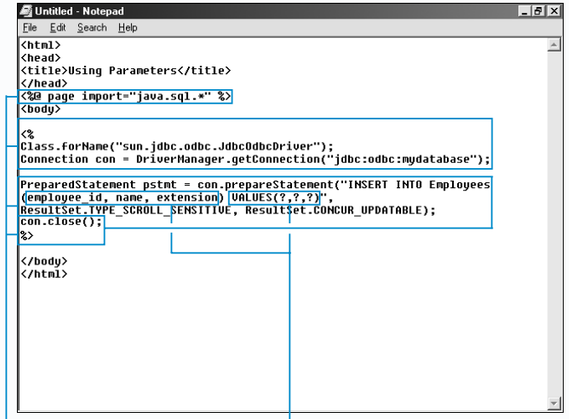
Once the values have been specified for the SQL statement, you can use the `execute` method of the `PreparedStatement` object to process the SQL statement using the parameters you set.

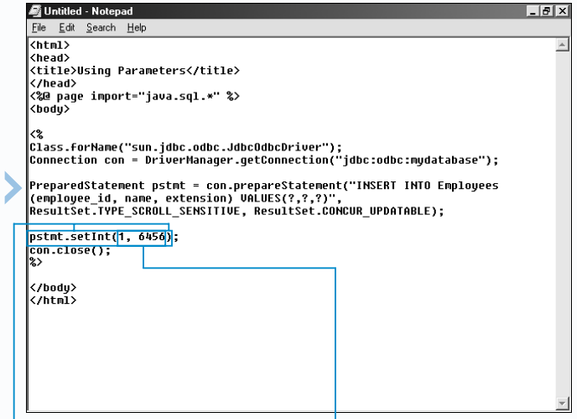
Extra

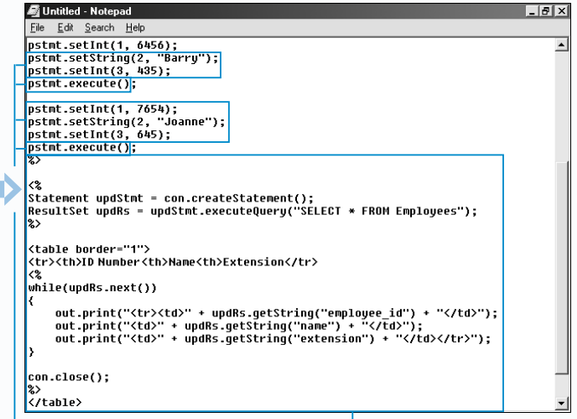
The following table displays the set methods commonly used to specify parameter values for specific data types. For a complete list of set methods and data types that can be used with prepared statements, refer to the `java.sql` package documentation.

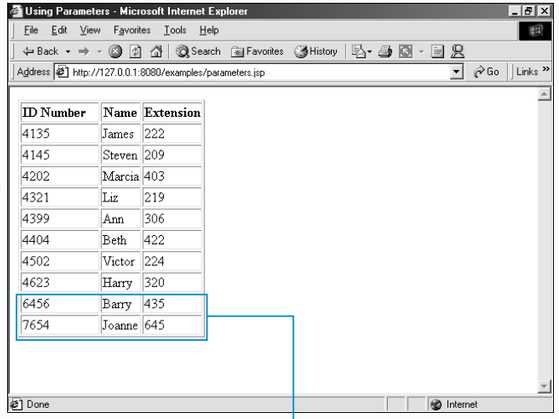
METHOD:	DATA TYPE:	METHOD:	DATA TYPE:
`setArray`	Array	`setFloat`	Float value
`setBigDecimal`	Large decimal number	`setInt`	Integer value
`setBlob`	Database blob type	`setLong`	Long value
`setBoolean`	Boolean value	`setNull`	Null value
`setByte`	Byte value	`setObject`	Object
`setBytes`	Array of bytes	`setShort`	Short value
`setDate`	Date	`setString`	String value
`setDouble`	Double value	`setTime`	Time

USING PARAMETERS IN A PREPARED STATEMENT









1 Type the code that connects the JSP page to the database to which you want to send a prepared statement.

2 Type the code that creates the `PreparedStatement` object and allows you to update the database.

3 Between the parentheses of the `prepareStatement` method, type the names of the fields in the table to which you want to add information. Separate each name with a comma.

4 Type **VALUES(?, ?, ?)**.

5 To create a method that will store a value for a question mark, type the name for the `PreparedStatement` object followed by a dot. Then type the set method you want to use followed by **()**.

6 Between the parentheses, type the number that indicates the position of the question mark in the SQL statement, followed by a comma and the value for the question mark.

■ String values must be enclosed in quotation marks.

7 Repeat steps 5 and 6 for each parameter you want to specify.

8 To execute the prepared statement, type the name for the `PreparedStatement` object followed by **.execute()**.

9 Type the code that retrieves the data from the database and formats the data for display.

10 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of using parameters in a prepared statement. The information in the database is also updated.

CALL A STORED PROCEDURE

A stored procedure is a set of instructions that are stored on a database server. A stored procedure can be as simple as an SQL statement that returns all the information in a table, but stored procedures are most often used to increase the efficiency of performing complex queries on a database. For example, stored procedures are ideal for tasks such as retrieving information based on a number of parameters. Using stored procedures tends to be more efficient than repeatedly using complex SQL statements because the stored procedures are compiled and executed within the database engine itself.

In order to use a stored procedure, the database program must support stored procedures and the stored procedure must be saved on the database server. Stored procedures are usually supported by large database programs such as Microsoft SQL Server and Oracle. Smaller database programs such as Microsoft Access often do not support stored procedures. You should consult the documentation included with your software to determine whether your database program supports stored procedures.

Before calling a stored procedure, you must first create a `CallableStatement` object that will retrieve the stored procedure from the database server. `CallableStatement` objects are commonly named cstmt. To create a `CallableStatement` object, you use the `prepareCall` method of the `Connection` object created when the database connection was set up. When using the `prepareCall` method, you use the `call` keyword followed by the name of the stored procedure you want to call. The name must match the name of a stored procedure already saved on the database server.

Once the `CallableStatement` object is created, the `executeQuery` method is used to generate a `ResultSet` object that will contain the results generated by the database using the stored procedure.

Extra

You can create a stored procedure by using a JSP page to issue SQL commands to the database server. You use the `execute` method of the `Statement` object to issue the SQL statements to the database server. When naming a stored procedure, you can use a lowercase first letter and then capitalize the first letter of each of the following words to make the name easy to read. For example, a stored procedure used to retrieve records in an employee database with extension numbers of more than three digits may be called getLargeExt.

Example:

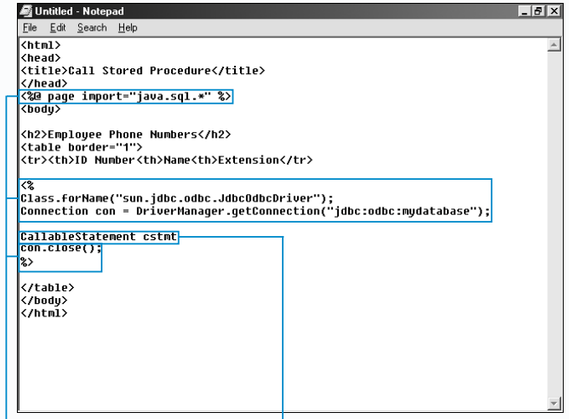
```
Statement stmt = con.createStatement();
stmt.execute("CREATE PROCEDURE getLargeExt AS
SELECT * FROM Employees WHERE extension > 999")
```

You can create a stored procedure directly on your database server. The SQL statements and methods you use to create a stored procedure on a database server depend on the database program you are using. You should consult your database program's documentation for information.

Example:

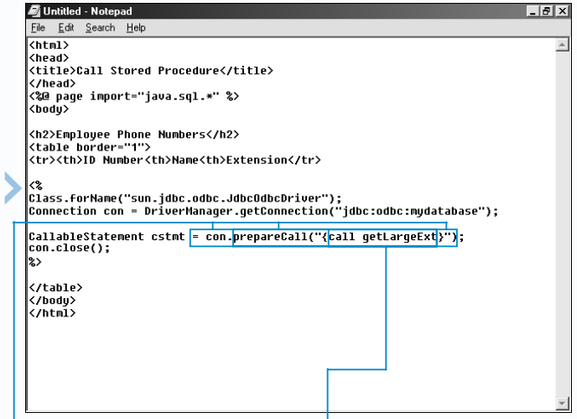
```
CREATE PROCEDURE getLargeExt AS SELECT * FROM Employees
WHERE extension > 999
```

CALL A STORED PROCEDURE



1 Type the code that creates the connection to the database that contains the stored procedure.

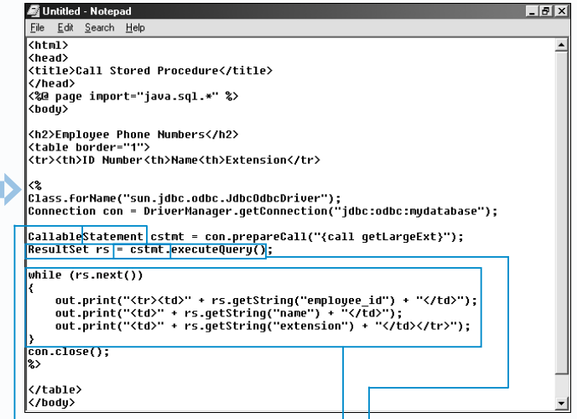
2 To create the `CallableStatement` object that will allow you to call a stored procedure, type **CallableStatement** followed by a name for the `CallableStatement` object.



3 Type **=** and the name of the `Connection` object followed by a dot.

4 Type **prepareCall**("{}").

5 Between the braces, type **call** followed by the name of the stored procedure you want to use.

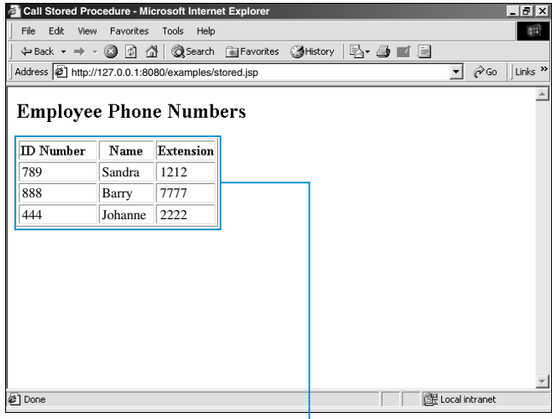


6 To create a `ResultSet` object to store the results returned from the database, type **ResultSet** followed by a name for the `ResultSet` object.

7 Type **=** and the name of the `CallableStatement` object followed by dot.

8 Type **executeQuery**().

9 Type the code that uses the results of the stored procedure.



10 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of calling a stored procedure.

GET DATABASE INFORMATION

The `DatabaseMetaData` object allows you to determine information about a database. Information you can determine using the `DatabaseMetaData` object includes a database program's configuration, the features the database supports and information about data stored in the database.

To determine information about a database, you first create a `DatabaseMetaData` object using the `getMetaData` method of the `Connection` object created when the connection was set up.

Once you create a `DatabaseMetaData` object, there are several methods of the object that you can use to determine specific information about the database to which you are connected.

One common use of the `DatabaseMetaData` object is to determine if a specific stored procedure exists on a database server. Stored procedures allow you to perform efficient queries on a database by storing and executing the instructions for the queries on the database server itself. You use the `getProcedures` method of the `DatabaseMetaData` object to retrieve the names of stored procedures available to the JSP page connected to the database. Using three `null` values as the arguments of the `getProcedures` method will retrieve a list of all the available stored procedures.

The information returned from the database using the `DatabaseMetaData` object is usually stored in a result set. Once information has been retrieved from a database and placed in a result set, you can retrieve the data from the result set. For information about retrieving information from a result set, see page 150.

Not all databases or database drivers will support all of the methods available to the `DatabaseMetaData` object. Typically, if a database or a database driver does not support a method implemented by the `DatabaseMetaData` object, an exception error will be generated. For information about error handling, see page 174.

GET DATABASE INFORMATION

Untitled - Notepad

```
<html>
<head>
<title>Available Stored Procedures</title>
</head>
<%@ page import="java.sql.*" %>
<body>

<h2>Available Stored Procedures</h2>
<ul>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:mydatabase");

DatabaseMetaData dbMeta = con.getMetaData();
ResultSet rs = dbMeta.getProcedures(null, null, null);

while(rs.next())
out.print("<li>" + rs.getString(3));

con.close();
%>
</ul>
</body>
</html>
```

Available Stored Procedures - Microsoft Internet Explorer

Available Stored Procedures

- getALL;1
- getBig;1
- getExt;1
- getLarge;1
- getNow;1

1 To retrieve information about the database to which a connection has been created, type **DatabaseMetaData** followed by a name for the `DatabaseMetaData` object.

2 Type **=** and the name of the `Connection` object followed by **.getMetaData()**.

3 To create a result set, type **ResultSet** followed by a name for the `ResultSet` object.

4 Type **=** and the name of the `DatabaseMetaData` object followed by **.getProcedures()**. Between the parentheses, type the arguments for the method.

5 Save the page with the .jsp extension and then display the JSP page in a Web browser.

The Web browser displays the result of retrieving information about a database.

COMMONLY USED METHODS OF THE `DatabaseMetaData` OBJECT

There are several methods of the `DatabaseMetaData` object that you can use to determine information about a database. For a complete list of the methods supported by the `DatabaseMetaData` object, consult the `java.sql` package documentation. Before using any of the following methods in your JSP code, you should check your database program's documentation to verify whether the program supports the method you want to use.

METHOD:	DATA TYPE:
`boolean` **`allProceduresAreCallable()`**	Determines whether a user can call all the procedures returned by the `getProcedures` method.
`ResultSet` **`getCatalogs()`**	Returns the catalog names that the database contains.
`Connection` **`getConnection()`**	Returns the ID of the connection that produced the `DatabaseMetaData` object.
`String` **`getDatabaseProductName()`**	Returns the name of the database program.
`String` **`getDatabaseProductVersion()`**	Returns the version number of the database program.
`String` **`getDriverVersion()`**	Returns the version number of the JDBC driver.
`int` **`getMaxColumnNameLength()`**	Returns the maximum length allowed for column headings.
`int` **`getMaxConnections()`**	Returns the maximum number of active connections the database can support at one time.
`int` **`getMaxRowSize()`**	Returns the maximum length allowed for a row.
`int` **`getMaxStatementLength()`**	Returns the maximum length allowed for an SQL statement.
`ResultSet` **`getProcedures`**`(String catalog, String schemaPattern, String procedureNamePattern)`	Returns the stored procedures available in the database.
`String` **`getSQLKeywords()`**	Returns a comma-separated list of all the SQL keywords from the database.
`ResultSet` **`getTableTypes()`**	Returns the table types available in the database.
`String` **`getUserName()`**	Returns the user name used to access the database.
`boolean` **`isReadOnly()`**	Indicates whether the database is in read-only mode.
`boolean` **`supportsBatchUpdates()`**	Indicates whether the driver supports batch updates.
`boolean` **`supportsMultipleResultSets()`**	Indicates whether you can create multiple result sets at once.
`boolean` **`supportsNonNullableColumns()`**	Indicates whether you can specify that columns must contain data.
`boolean` **`supportsOuterJoins()`**	Indicates whether outer joins are supported.
`boolean` **`supportsStoredProcedures()`**	Indicates whether you can use stored procedures with the database.
`boolean` **`usesLocalFiles()`**	Indicates whether the database stores tables in a local file.

USING A JAVABEAN TO ACCESS A DATABASE

Accessing a database from a JSP page requires large amounts of Java code. You can use a JavaBean to separate the code that performs this task from the HTML code in the JSP page.

The code required to create a JavaBean that accesses a database is similar to that used to access a database directly from a JSP page. To retrieve information from a database, you must set up a connection to the database and then create a result set to store the retrieved information.

You can create a constructor method to perform initialization tasks in the JavaBean. A constructor method has the same name as the class and is executed when the JavaBean is instantiated. The constructor method for a JavaBean that accesses a database may load the appropriate drivers and connect to the database. You use the `Class.forName` statement to specify the name of the driver you want to load. A `Connection` object in the constructor method allows the JavaBean to connect to the database. The constructor method should also include the code to retrieve information from the database and store the information in a result set.

Retrieving information from a result set must be done using JavaBean properties. You can create a property and getter method for each column in the database you want to be made available through the JavaBean. For more information about getter methods, see page 128. You should declare the `ResultSet` object and properties in the body of the class. If you declare the object and properties elsewhere, such as within a method of the class, the getter methods may not be able to access the result set or the properties.

When using a JavaBean to access a database, errors may occur. You should ensure that the JavaBean includes `try` blocks and `catch` blocks to handle any errors.

Apply It

To use a JavaBean you created to access a database, you use the `property` attribute of the `<jsp:getProperty>` tag to specify which column in the database you want to access. The JavaBean will then make the connection to the database and use the appropriate getter method of the JavaBean required to retrieve the information.

TYPE THIS:

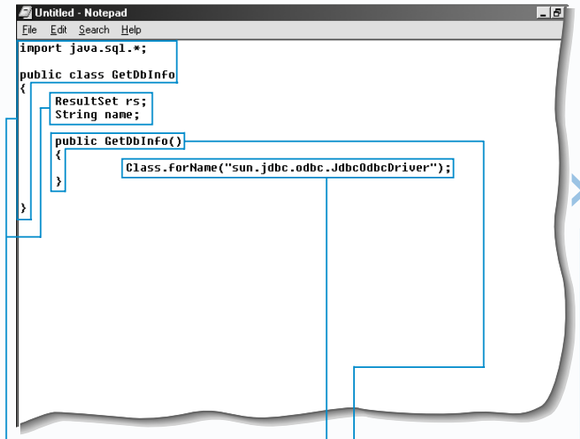
```
<html>
<head>
<jsp:useBean id="DbBean" scope="session" class="GetDbInfo" />
<jsp:setProperty name="DbBean" property="*" />
</head>
<body>
The first three names in the database are:
<ul>
<%
for (int x = 0; x < 3; x++)
{
%>
      <li><jsp:getProperty name="DbBean" property="name" /></li>
<%
}
%>
</ul>
</body>
</html>
```

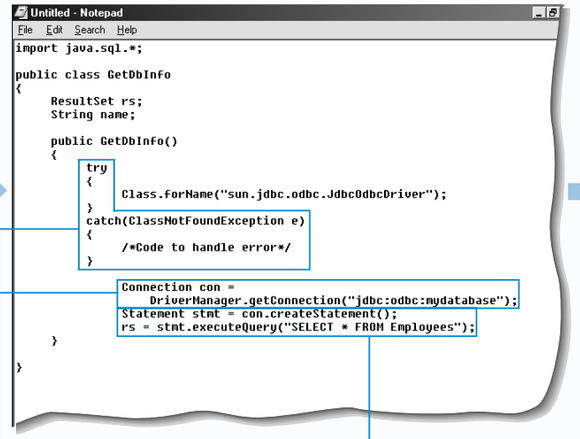
RESULT:

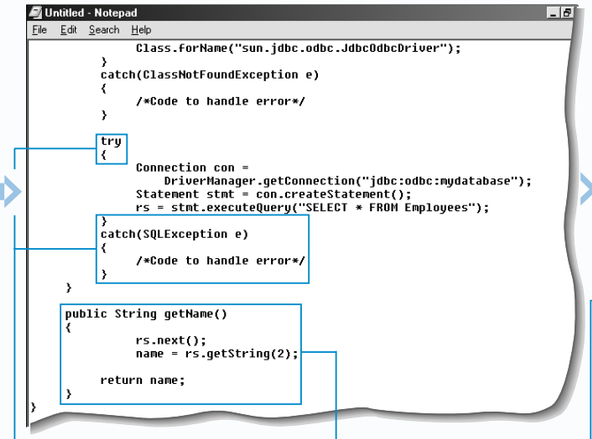
The first three names in the database are:

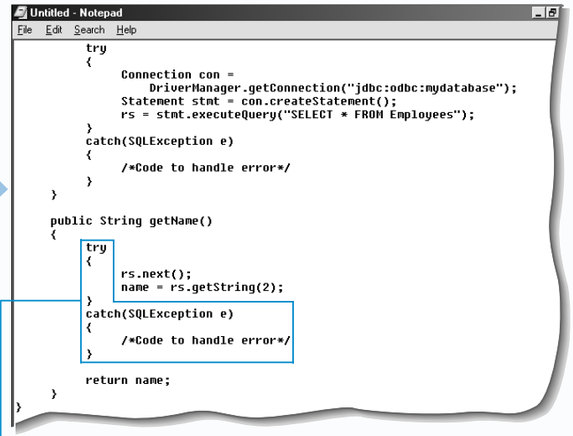
- James
- Steven
- Marcia

USING A JAVABEAN TO ACCESS A DATABASE









1 Type the code that imports the `java.sql` package and creates a JavaBean class.

2 Type the code that declares a `ResultSet` object and declares a property for each column in the database you want to be able to access from a JSP page.

3 Type the code that creates the constructor method for the JavaBean.

4 Type the code that loads the bridge driver.

5 Create a `try` block and a `catch` block that will handle any exceptions that may be thrown when loading the bridge driver.

6 Type the code that creates a `Connection` object that specifies the location of the database you want to connect to.

7 Type the code that retrieves information from the database and stores it in a result set.

8 Create a `try` block and a `catch` block that will handle any exceptions that may be thrown when retrieving information from the database.

9 Type the code that creates the getter method that will retrieve information from the record set and return the value of a property specified in step 2.

10 Create a `try` block and a `catch` block that will handle any exceptions that may be thrown while accessing the database.

11 Save the file with the .java extension and then compile the source code for the file.

12 Copy the compiled class file to the appropriate directory on your Web server.

- You can now use the JavaBean to access a database in your JSP pages.