Accessing Databases using JDBC

Overview

In this exercise, you will use JDBC to create a table in a relational database, insert data, query data, and interrogate the database via metadata.

You will perform the following tasks:

- Load the appropriate database driver for the Derby database.
- Open a connection on the database.
- Create `Statement` objects and `PreparedStatement` objects as appropriate.
- Execute SQL Data Definition Language (DDL) and Data Manipulation Language (DML) statements on the database.
- Where appropriate, process the results of SQL queries by using a `ResultSet`.
- Optionally, retrieve metadata for a `ResultSet` to find out information about the structure and data types of the returned columns.

Source folders

Student project: `StudentJDBC`
Solution project: `SolutionJDBC`

You will use the `C:/JavaDev/Databases/MyDatabase` sample database, i.e. the same as during the demos in the chapter.

Roadmap

All of the exercises in this lab use the sample Derby database. Before you go any further, start the Derby database by running the `C:\JavaDev\Databases\StartDerby.bat` batch file.

There are four exercises in this lab, of which the final exercise is optional. Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Load the Derby JDBC driver, and open a connection to the aforementioned database. You will also implement the `createTable()` method; this method will execute a SQL `CREATE TABLE` statement, to create a new table named `MySchema.Books` in the database.

2. Implement the `queryData()` method. This method will execute a SQL `SELECT` statement, to retrieve and display information for all entries in the `MySchema.Books` table.

3. Implement the `insertData()` method. This method will execute a series of SQL `INSERT` statements, to insert new books into the `MySchema.Books` table. For efficiency, you will use a `PreparedStatement` to perform these insertions.

4. If time permits, implement the `executeRawSQL()` method. This method will allow the user to enter any SQL from the keyboard. You will execute this SQL statement, and then extract metadata to process the results.

Exercise 1: Connect to the database, and create a new Books table

Open the student project, `StudentJDBC`, and take a look at `Main.java`.

In the `main()` method:

- Load the JDBC driver to allow you to work with Derby databases. Note that you must also configure the project classpath to include the JAR file for the Derby JDBC driver (i.e. `C:\JavaDev\Databases\Derby\lib\derbyclient.jar`).
- Then get a connection to the database; use the global `connection` variable to hold this connection.

Now implement `createTable()`, so that the user can create a `MySchema.Books` table in the test database. Here's a Java `String` variable that contains the SQL you need to execute; ask the instructor if you need clarification on what this means. You can copy this string from `SqlCreateTable.txt` file in the student folder, to save you some time and grief ☺.

```
String sql = "CREATE TABLE MySchema.Books ( " +
                "Isbn  INTEGER      NOT NULL, " +
                "Title VARCHAR(50)  NOT NULL, " +
                "Price DECIMAL(6,2) NOT NULL, " +
                "CONSTRAINT PK_Books PRIMARY KEY (Isbn) "    +
             ")";
```

Run the application. In the *Console* window at the bottom of the Eclipse IDE, verify that the application displays the list of options. Select option 1 to create the `MySchema.Books` table.

Exercise 2: Query the Books table

Implement the `queryData()` method, so that it gets the `Isbn`, `Title`, and `Price` for every book in the `MySchema.Books` table. Display the details for each book on the console window.

Run your application when you are ready. Select option 2 to query the `MySchema.Books` table. It should be empty at the moment.

Exercise 3: Insert new books into the Books table

Implement the `insertData()` method, so that it inserts a series of new books into the `MySchema.Books` table. Here are some hints:

- Create a `PreparedStatement` object containing the following SQL:
  `"INSERT INTO MySchema.Books VALUES (?, ?, ?)"`
- Set up a loop that gives the user an opportunity to enter data for many new books. On each loop iteration, ask the user to enter the isbn, title, and price for the new book. Convert the isbn into an `int`; leave the title as a `String`; and convert the price into a `java.math.BigDecimal`.
- Assign these values to the parameters in the `PreparedStatement`, and then execute `PreparedStatement`.
- Test the result of the insertion, to verify that the new row has been inserted successfully.

Run your application. Select option 3 to insert some new rows into the `MySchema.Books` table. Then select option 2, to ensure that the new rows have indeed been inserted.

Exercise 4 (If Time Permits): Execute raw SQL and use metadata to see the effect

Implement the `executeRawSQL()` method, so that it accepts a string of raw SQL from the user and executes it. Here are some guidelines; we'll leave you to figure out the details:

- Ask the user to enter a string of SQL.
- Execute the SQL statement.
- The `Statement` class has a `getResultSet()` method to determine whether the statement returned a result set (i.e. it was a `SELECT` statement). In this case:
  - Get metadata for the `ResultSet`.
  - Display the name, type, and class name of each column in the `ResultSet`; see the `ResultSetMetaData` class for a list of the APIs available to you.
  - Also iterate through the `ResultSet` itself, and display the data in each row in the `ResultSet`.
- If the statement didn't return a `ResultSet`, you can call `getUpdateCount()` to determine the number of rows affected by the SQL statement.

Test the application thoroughly with different kinds of SQL statement.