Java Tutorial

An Introduction to Java Database (JDBC) Programming by Examples

TABLE OF CONTENTS (HIDE)

1. Relational Database and and Structure Query Language (SQL)

I presume that you have some knowledge on Relational Databases and the SQL language. Otherwise, read "Introduction to Relational Database and SQL".

2. Setting-up MySQL

In this section, I shall describe the MySQL relational database system. For other database systems, read "Other Databases".

Install MySQL (read "How to Set Up MySQL and Get Started"). I shall assume that the MySQL server is running on the default port number 3306.

On MySQL, instead of using the "`mysql`" interactive *client* program provided, you can write your own *client* programs (in Java or other languages) to access the MySQL server. Your client programs shall connect to the database server at the given IP address and TCP port number, issue the SQL commands, and process the results received.

2.1 Install JDK and Programming Editor

Before you proceed, I shall assume that you are familiar with Java Programming and have installed the followings:

1. JDK (Read "How to install JDK and Get Started").
2. A programming text editor, such as TextPad, Notepad++, Sublime, Atom for Windows (Read "Programming Editors for Windows"), or jEdit, gEdit, Sublime, Atom for Mac OS (Read "Programming Editors for Mac OS").

2.2 Install MySQL JDBC Driver (Don't MISS This Step!)

You need to install an appropriate JDBC (Java Database Connectivity) driver to run your Java database programs. The MySQL's JDBC driver is called "MySQL Connector/J" and is available at MySQL mother site.

For Windows

1. Download the "latest" MySQL JDBC driver from http://dev.mysql.com/downloads ⇒ "MySQL Connectors" ⇒ "Connector/J" ⇒ Connector/J 5.1.{*xx*} ⇒ select "Platform Independent" ⇒ ZIP Archive (e.g., "`mysql-connector-java-5.1.{xx}.zip`", where {*xx*} is the latest release number) ⇒ "No thanks, just start my download".
2. UNZIP the download file into a temporary folder.
3. From the temporary folder, COPY the JAR file "`mysql-connector-java-5.1.{xx}-bin.jar`" to your JDK's Extension Directory at "`<JAVA_HOME>\jre\lib\ext`" (where `<JAVA_HOME>` is the JDK installed directory), e.g., "`c:\program files\java\jdk1.8.0_{xx}\jre\lib\ext`".

For Mac OS

1. Download the latest MySQL JDBC driver from http://www.mysql.com/downloads ⇒ MySQL Connectors ⇒ Connector/J ⇒ Connector/J 5.1.{*xx*} ⇒ select "Platform Independent" ⇒ Compressed TAR Archive (e.g., `mysql-connector-java-5.1.{xx}.tar.gz`, where {*xx*} is the latest release number).

2. Double-click on the downloaded TAR file to expand into folder "`mysql-connector-java-5.1.{xx}`".
3. Open the expanded folder. COPY the JAR file "`mysql-connector-java-5.1.{xx}-bin.jar`" to JDK's extension directory at "`/Library/Java/Extensions`" (To goto that folder, from "Finder" ⇒ GO ⇒ Go to Folder ⇒ Type "`/Library/Java/Extensions`").

(For Advanced User Only) You can compile Java database programs without the JDBC driver. But to run the JDBC programs, the JDBC driver's JAR-file must be included in the environment variable `CLASSPATH`, or the JDK's extension directory, or in the `java`'s command-line option `-cp <paths>`.

You can set the `-cp` option for Java runtime as follows:

```
// For windows
java -cp .;/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun
// For Mac OS/Unixes
java -cp .:/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun
```

2.3 Setup Database

We have to set up a database before embarking on our database programming. We shall call our database "ebookshop" which contains a table called "books", with 5 columns, as below:

```
Database: ebookshop
Table: books
+-------+----------------------------+---------------+---------+-------+
| id    | title                      | author        | price   | qty   |
| (INT) | (VARCHAR(50))              | (VARCHAR(50)) | (FLOAT) | (INT) |
+-------+----------------------------+---------------+---------+-------+
| 1001  | Java for dummies           | Tan Ah Teck   | 11.11   |   11  |
| 1002  | More Java for dummies      | Tan Ah Teck   | 22.22   |   22  |
| 1003  | More Java for more dummies | Mohammad Ali  | 33.33   |   33  |
| 1004  | A Cup of Java              | Kumar         | 44.44   |   44  |
| 1005  | A Teaspoon of Java         | Kevin Jones   | 55.55   |   55  |
+-------+----------------------------+---------------+---------+-------+
```

Start MySQL Server: Start the MySQL server and verify the server's TCP port number from the console messages. I shall assume that MySQL server is running on the default port number of 3306.

```
// For Windows
cd {path-to-mysql-bin}   // Check your MySQL installed directory
mysqld --console

// For Mac OS
// Use graphical control at "System Preferences" -> MySQL -> Start|Stop
```

Start a MySQL client: I shall also assume that there is an authorized user called "`myuser`" with password "xxxx".

```
// For Windows
cd {path-to-mysql-bin}   // Check your MySQL installed directory
mysql -u myuser -p

// For Mac OS
cd /usr/local/mysql/bin
./mysql -u myuser -p
```

Run the following SQL statements to create our test database and table.

```
create database if not exists ebookshop;

use ebookshop;

drop table if exists books;
create table books (
   id int,
   title varchar(50),
   author varchar(50),
   price float,
   qty int,
   primary key (id));

insert into books values (1001, 'Java for dummies', 'Tan Ah Teck', 11.11, 11);
insert into books values (1002, 'More Java for dummies', 'Tan Ah Teck', 22.22, 22);
insert into books values (1003, 'More Java for more dummies', 'Mohammad Ali', 33.33, 33);
insert into books values (1004, 'A Cup of Java', 'Kumar', 44.44, 44);
insert into books values (1005, 'A Teaspoon of Java', 'Kevin Jones', 55.55, 55);
```

```
select * from books;
```

3. Introduction to JDBC Programming by Examples

A JDBC (Java Database Connectivity) program comprises the following steps:

1. Allocate a `Connection` object, for connecting to the database server.
2. Allocate a `Statement` object, under the `Connection` object created, for holding a SQL command.
3. Write a SQL query and execute the query, via the `Statement` and `Connection` created.
4. Process the query result.
5. Close the `Statement` and `Connection` object to free up the resources.

We shall illustrate Java Database programming by the following examples.

IMPORTANT: The following examples require JDK 1.7 and above to run. If you JDK is below 1.7, upgrade your JDK (See JDK How-to)! You can check your JDK version via command "`javac -version`".

3.1 Example 1: SQL `SELECT`

Try out the following JDBC program (requires JDK 1.7 and above), which issues an SQL `SELECT`. Take note that the *source filename* must be the *same* as the *classname*, with extension of "`.java`". Save the program in any directory of your choice (e.g., `d:/myproject`).

```
1  import java.sql.*;   // Use 'Connection', 'Statement' and 'ResultSet' classes in java.sql package
2
3  // JDK 1.7 and above
4  public class JdbcSelectTest {   // Save as "JdbcSelectTest.java"
5     public static void main(String[] args) {
6        try (
7           // Step 1: Allocate a database 'Connection' object
8           Connection conn = DriverManager.getConnection(
9                 "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx");
10                // MySQL: "jdbc:mysql://hostname:port/databaseName", "username", "password"
11
12          // Step 2: Allocate a 'Statement' object in the Connection
13          Statement stmt = conn.createStatement();
14       ) {
15          // Step 3: Execute a SQL SELECT query, the query result
16          //  is returned in a 'ResultSet' object.
17          String strSelect = "select title, price, qty from books";
18          System.out.println("The SQL query is: " + strSelect); // Echo For debugging
19          System.out.println();
20
21          ResultSet rset = stmt.executeQuery(strSelect);
22
23          // Step 4: Process the ResultSet by scrolling the cursor forward via next().
24          //  For each row, retrieve the contents of the cells with getXxx(columnName).
25          System.out.println("The records selected are:");
26          int rowCount = 0;
27          while(rset.next()) {   // Move the cursor to the next row, return false if no more row
28             String title = rset.getString("title");
29             double price = rset.getDouble("price");
30             int    qty   = rset.getInt("qty");
31             System.out.println(title + ", " + price + ", " + qty);
32             ++rowCount;
33          }
34          System.out.println("Total number of records = " + rowCount);
35
36       } catch(SQLException ex) {
37          ex.printStackTrace();
38       }
39       // Step 5: Close the resources - Done automatically by try-with-resources
40    }
41 }
```

Compile: To compile the program:

```
cd path-to-the-java-source-file
javac JdbcSelectTest.java
```

Run: To run the program:

```
java JdbcSelectTest
```

(Skip Unless...) Read "Common Errors in JDBC Programming on MySQL".

Dissecting the Program

1. The JDBC operations are carried out through the "`Connection`", "`Statement`" and "`ResultSet`" objects (defined in package `java.sql`). However, you need not know the details, but merely the public methods defined in the API (Application Program Interface). You also need not re-invent the wheels by creating these classes yourself (which will take you many years?!). "Re-using" software component is a main strength of OOP.
2. Notice that there is little programming involved in using JDBC programming. You only have to specify the *database-URL*, write the SQL query, and process the query result. The rest of the codes are kind of "standard JDBC program template". Again, this is because the wheels have been invented.
3. In Line 8, we allocate a `Connection` object (called `conn`) via `static` method `DriverManager.getConnection(`*`database-url, db-user, password`*`)`. The Java program uses a so-called *database-URL* to connect to the server:
   - For MySQL:

     ```
     // Syntax
     Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:{port}/{db-name}", "{db-user}", "{password}");
     // Example
     Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ebooksop", "myuser", "xxxx");
     ```

     The `database-url` is in the form of "`jdbc:mysql://{host}:{port}/{database-name}`", with protocol `jdbc` and sub-protocol `mysql`. The `port` specifies the MySQL server's TCP port number; `db-user/password` is an authorized MySQL user. In our example, "`localhost`" (with special IP address of `127.0.0.1`) is the hostname for local loop-back; "`3306`" is the server's TCP port number, and ebookshop is the database name.
   - Others: Read "Other Databases".
4. In Line 13, we allocate a `Statement` object (called `stmt`) within the `Connection` via `conn.createStatement()`.
5. To execute a SQL SELECT command, we invoke method `stmt.executeQuery("SELECT ...")`. It returns the query result in a `ResultSet` object (called `rset`). `ResultSet` models the returned table, which can be access via a *row cursor*. The cursor initially positions before the first row in the `ResultSet`. `rset.next()` moves the cursor to the first row. You can then use `rset.getXxx(`*`columnName`*`)` to retrieve the value of the column for that row, where `Xxx` corresponds to the type of the column, such as `int`, `float`, `double` and `String`. The `rset.next()` returns `false` at the last row, which terminates the while-loop.
6. You may use `rset.getString(`*`columnName`*`)` to retrieve all types (`int`, `double`, etc).
7. For maximum portability, `ResultSet` columns within each row should be read in left-to-right order, and *each column should be read only once* via the `getXxx()` methods. Issue `getXxx()` to a cell more than once may trigger a strange error.
8. In this example, we use JDK 1.7's new feature called try-with-resources, which automatically closes all the opened resources in the try-clause, in our case, the `Connection` and `Statement` objects.

Exercises: Modify your Java program to issue the following `SELECT` statements and display all the columns retrieved. Make sure you modify the `ResultSet` processing to process only the columns retrieved (otherwise, you will get a "Column not found" error).

1. `SELECT * FROM books`
2. `SELECT title, author, price, qty FROM books WHERE author = 'Tan Ah Teck' OR price >= 30 ORDER BY price DESC, id ASC`

3.2 Example 2: SQL UPDATE

To execute a SQL `UPDATE`, you have to invoke the method `executeUpdate()` of the `Statement` object, which returns an `int` indicating the number of records affected. Recall that for `SELECT`, we use `executeQuery()`, which returns a `ResultSet` object modeling the returned table. SQL `UPDATE|INSERT|DELETE` does not return a table, but an `int` indicating the number of records affected.

```
1  import java.sql.*;     // Use classes in java.sql package
2
3  // JDK 1.7 and above
4  public class JdbcUpdateTest {    // Save as "JdbcUpdateTest.java"
5     public static void main(String[] args) {
6        try (
7           // Step 1: Allocate a database 'Connection' object
8           Connection conn = DriverManager.getConnection(
9                 "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx"); // MySQL
10
11          // Step 2: Allocate a 'Statement' object in the Connection
12          Statement stmt = conn.createStatement();
13       ) {
14          // Step 3 & 4: Execute a SQL UPDATE via executeUpdate()
15          //   which returns an int indicating the number of rows affected.
```

```
         // Increase the price by 7% and qty by 1 for id=1001
         String strUpdate = "update books set price = price*0.7, qty = qty+1 where id = 1001";
         System.out.println("The SQL query is: " + strUpdate);  // Echo for debugging
         int countUpdated = stmt.executeUpdate(strUpdate);
         System.out.println(countUpdated + " records affected.");

         // Step 3 & 4: Issue a SELECT to check the UPDATE.
         String strSelect = "select * from books where id = 1001";
         System.out.println("The SQL query is: " + strSelect);  // Echo for debugging
         ResultSet rset = stmt.executeQuery(strSelect);
         while(rset.next()) {   // Move the cursor to the next row
            System.out.println(rset.getInt("id") + ", "
                    + rset.getString("author") + ", "
                    + rset.getString("title") + ", "
                    + rset.getDouble("price") + ", "
                    + rset.getInt("qty"));
         }
      } catch(SQLException ex) {
         ex.printStackTrace();
      }
      // Step 5: Close the resources - Done automatically by try-with-resources
   }
}
```

Exercises: Modify your Java program to issue the following SQL statements:

1. Increase the `price` by 50% for "A Cup of Java".
2. Set the `qty` to 0 for "A Teaspoon of Java".

3.3 Example 3: SQL `INSERT` and `DELETE`

Similarly, use the `executeUpdate()` to execute `'INSERT INTO'` and `'DELETE FROM'`. The method returns an `int` indicating the number of records affected.

```
import java.sql.*;     // Use classes in java.sql package

// JDK 1.7 and above
public class JdbcInsertTest {    // Save as "JdbcUpdateTest.java"
   public static void main(String[] args) {
      try (
         // Step 1: Allocate a database 'Connection' object
         Connection conn = DriverManager.getConnection(
               "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx"); // MySQL

         // Step 2: Allocate a 'Statement' object in the Connection
         Statement stmt = conn.createStatement();
      ) {
         // Step 3 & 4: Execute a SQL INSERT|DELETE statement via executeUpdate(),
         //   which returns an int indicating the number of rows affected.

         // DELETE records with id>=3000 and id<4000
         String sqlDelete = "delete from books where id>=3000 and id<4000";
         System.out.println("The SQL query is: " + sqlDelete);  // Echo for debugging
         int countDeleted = stmt.executeUpdate(sqlDelete);
         System.out.println(countDeleted + " records deleted.\n");

         // INSERT a record
         String sqlInsert = "insert into books " // need a space
               + "values (3001, 'Gone Fishing', 'Kumar', 11.11, 11)";
         System.out.println("The SQL query is: " + sqlInsert);  // Echo for debugging
         int countInserted = stmt.executeUpdate(sqlInsert);
         System.out.println(countInserted + " records inserted.\n");

         // INSERT multiple records
         sqlInsert = "insert into books values "
               + "(3002, 'Gone Fishing 2', 'Kumar', 22.22, 22),"
               + "(3003, 'Gone Fishing 3', 'Kumar', 33.33, 33)";
         System.out.println("The SQL query is: " + sqlInsert);  // Echo for debugging
         countInserted = stmt.executeUpdate(sqlInsert);
         System.out.println(countInserted + " records inserted.\n");

         // INSERT a partial record
         sqlInsert = "insert into books (id, title, author) "
               + "values (3004, 'Fishing 101', 'Kumar')";
         System.out.println("The SQL query is: " + sqlInsert);  // Echo for debugging
```

```
42            countInserted = stmt.executeUpdate(sqlInsert);
43            System.out.println(countInserted + " records inserted.\n");
44
45            // Issue a SELECT to check the changes
46            String strSelect = "select * from books";
47            System.out.println("The SQL query is: " + strSelect);  // Echo For debugging
48            ResultSet rset = stmt.executeQuery(strSelect);
49            while(rset.next()) {   // Move the cursor to the next row
50               System.out.println(rset.getInt("id") + ", "
51                       + rset.getString("author") + ", "
52                       + rset.getString("title") + ", "
53                       + rset.getDouble("price") + ", "
54                       + rset.getInt("qty"));
55            }
56         } catch(SQLException ex) {
57            ex.printStackTrace();
58         }
59         // Step 5: Close the resources - Done automatically by try-with-resources
60      }
61   }
```

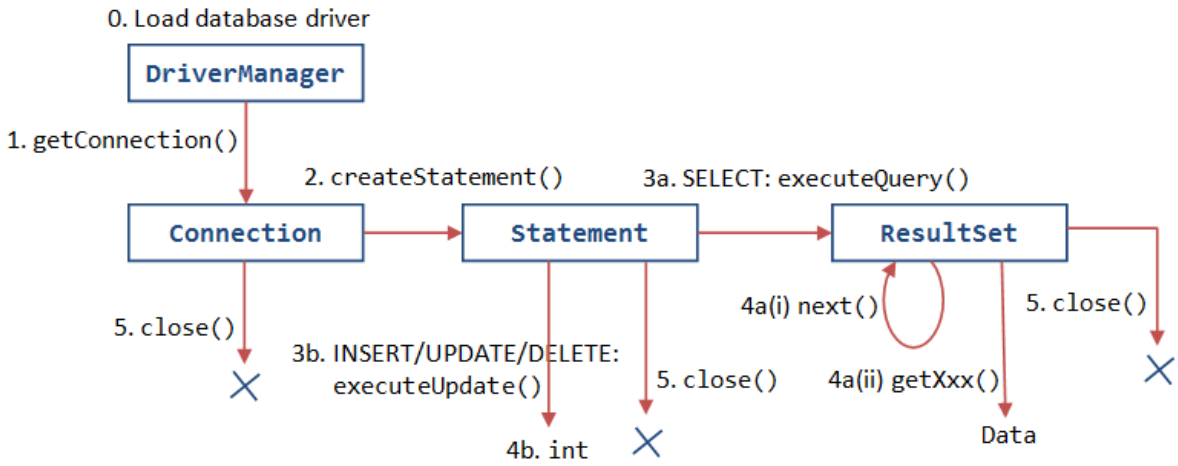
Notes:

1. You cannot insert two records with the same primary key (i.e., `id`) value. Hence, we issue a `DELETE` before `INSERT` new record. In this way, you can re-run the program.
2. If you insert a partial record, the missing columns will be set to their default values.

Exercise: Modify your Java program to issue the following SQL statements:

1. Delete all books with id > 8000; and insert: (8001, 'Java ABC', 'Kevin Jones', 15.55, 55) and (8002, 'Java XYZ', 'Kevin Jones', 25.55, 55);

4. JDBC Cycle



REFERENCES & RESOURCES

1. JDBC Online Tutorial @ http://download.oracle.com/javase/tutorial/jdbc/index.html.
2. JDBC Home Page @ http://java.sun.com/products/jdbc/overview.html.
3. JDBC API Specifications 1.2, 2.1, 3.0, and 4.0 @ http://java.sun.com/products/jdbc.
4. White Fisher, et al., "JDBC API Tutorial and Reference", 3rd eds, Addison Wesley, 2003.
5. MySQL Home Page @ http://www.mysql.org, and documentation.
6. MySQL 5.5 Reference Manual @ http://dev.mysql.com/doc/refman/5.5/en/index.html.
7. SQL.org @ http://www.sql.org.
8. Russell Dyer, "MySQL in a Nutshell", O'Reilly, 2008.

Latest version tested: Latest version tested: JDK 1.8.0_112/JDBC 4.0, MySQL 5.7.17, Connector/J 5.1.40
Last modified: January, 2017

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | HOME