Java Tutorial

An Introduction to Java Database (JDBC) Programming by Examples

TABLE OF CONTENTS (HIDE)

- 1. Relational Database and Structu
- 2. Setting-up MySQL
 - 2.1 Install JDK and Programming
 - 2.2 Create a Directory to Keep all
 - 2.3 Install MySQL JDBC Driver (Dc
 - 2.4 Setup Database
- 3. Introduction to JDBC Programm
 - 3.1 Example 1: SQL SELECT
 - 3.2 Example 2: SQL UPDATE
- 3.3 Example 3: SQL INSERT and D

▶

4. JDBC Cycle

1. Relational Database 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 Create a Directory to Keep all your Works

I shall assume that you have created a directory called "c:\myWebProject" (for Windows) or "~/myWebProject" (for Mac OS X) in your earlier exercises. Do it otherwise.

This step is important; otherwise, you will be out-of-sync with this article and will not be able to find your files later.

2.3 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 \Rightarrow "MySQL Connectors" \Rightarrow "Connector/J"

 \Rightarrow Connector/J 5.1.{xx} \Rightarrow select "Platform Independent" \Rightarrow <u>ZIP Archive</u> (e.g., "mysql-connector-java-5.1.{xx}.zip", where {xx} is the latest release number) \Rightarrow "No thanks, just start my download".

- 2. UNZIP the download file into your project directory "C:\myWebProject".
- 3. Open the unzipped folder. Check for JAR file "mysql-connector-java-5.1.{xx}-bin.jar". The absolute filename of the JAR file (which you will need later) is "C:\myWebProject\mysql-connector-java-5.1.{xx}\mysql-co

For Mac OS X

- 1. Download the latest MySQL JDBC driver from http://dev.mysql.com/downloads \Rightarrow MySQL Connector/J \Rightarrow Connector/J $5.1.\{xx\}$ \Rightarrow select "Platform Independent" \Rightarrow 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. Move the expanded folder into your project directory "~/myWebProject".
- 4. Open the expanded folder. Check for JAR file "mysql-connector-java-5.1.{xx}-bin.jar". The absolute filename of the JAR file (which you will need later) is "~/myWebProject/mysql-connector-java-5.1.{xx}/mysql-connector-java-5.1.{xx}-bin.jar". Take note of this super long filename that you will need later. COPY and SAVE in a scratch pad so that you can retrieve later.

(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 (prior to JDK 9), or in the java's command-line option -cp.

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

For JDK 8 and lower, you can copy the JAR file into JDK's extension directory at "<JAVA_HOME>\jre\lib\ext" (Windows) or "/Library/Java/Extensions" (Mac OS X).

JDK 9 remove the extension mechanism.

2.4 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 (INT)	title (VARCHAR(50))	author (VARCHAR(50))	+ price (FLOAT)	qty (INT)
1001 1002 1003 1004 1005	Java for dummies More Java for dummies More Java for more dummies A Cup of Java A Teaspoon of Java	Tan Ah Teck Tan Ah Teck Mohammad Ali Kumar Kevin Jones	11.11 22.22 33.33 44.44 55.55	11 22 33 44 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: assume that MySQL is installed in "c:\myWebProject\mysql"
c:

mysqld --console
// For Mac OS X
// 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: assume that MySQL is installed in "c:\myWebProject\mysql"
c:
cd \myWebProject\mysql\bin
mysql -u myuser -p
// For Mac OS X: assume that MySQL is installed in "/usr/local/mysql"
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;
```

cd \myWebProject\mysql\bin

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 into your project directory "c:\myWebProject" (Windows) or "~/myWebProject" (Mac OS X), as "JdbcSelectTest.java".

1 import java.sql.*; // Use 'Connection', 'Statement' and 'ResultSet' classes in java.sql package

2

```
// JDK 1.7 and above
3
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(
                    "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx");
9
                    // MySQL: "jdbc:mysql://hostname:port/databaseName", "username", "password"
10
11
              // Step 2: Allocate a 'Statement' object in the Connection
12
              Statement stmt = conn.createStatement();
13
           ) {
14
              // Step 3: Execute a SQL SELECT query, the query result
15
              // is returned in a 'ResultSet' object.
16
              String strSelect = "select title, price, qty from books";
17
18
              System.out.println("The SQL query is: " + strSelect); // Echo For debugging
              System.out.println();
19
20
21
              ResultSet rset = stmt.executeQuery(strSelect);
22
23
              // Step 4: Process the ResultSet by scrolling the cursor forward via next().
              // For each row, retrieve the contents of the cells with getXxx(columnName).
24
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");
                     qty = rset.getInt("qty");
30
                 int
                 System.out.println(title + ", " + price + ", " + qty);
31
32
                 ++rowCount;
              }
33
              System.out.println("Total number of records = " + rowCount);
34
35
           } catch(SQLException ex) {
36
37
              ex.printStackTrace();
38
           }
39
           // Step 5: Close the resources - Done automatically by try-with-resources
40
        }
41
    }
```

Compile: To compile the program:

```
// Windows: The Java source directory is "c:\myWebProject"
c:
cd \myWebProject
javac JdbcSelectTest.java
// Mac OS X: The Java source directory is "~/myWebProject"
cd ~/myWebProject
javac JdbcSelectTest.java
```

Run: It is rather difficult to run the program, as you need to include the MySQL JDBC Driver in the classpath (via - cp option) as follows:

```
// For windows
c:
cd \myWebProject
java -cp .;C:\myWebProject\mysql-connector-java-5.1.{xx}\mysql-connector-java-5.1.{xx}-bin.jar JdbcSelectTest
// For Mac OS X
cd ~/myWebProject
// Need to use $HOME instead of ~ in the "java" command
java -cp .:$HOME/myWebProject/mysql-connector-java-5.1.{xx}/mysql-connector-java-5.1.{xx}-bin.jar JdbcSelectTest
```

You should COPY and SAVE this command to your notepad, so that you don't need to type this super-long command again and again. (For Mac User, DO NOT use the default TextEdit to save this command!!! Use a programming text editor.)

Figure out that the part marked in RED is the absolute filename of the JAR file I asked you to take note earlier, when you installed the MySQL JDBC Driver.

This is a little messy here. The "." denotes the current directory to locate the JdbcSelectTest. The ";" or ":" are path separator for Windows and Mac OS X, respectively. In other words, there are 2 items in the classpath: the current directory and the MySQL JDBC Driver.

(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/ebookshop", "myuser", "xxxx");
```

The database-url is in the form of "jdbc:mysql://{host}:{port}/{database-name}", with protocol jdbc and subprotocol 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.

Save the program into your project directory "c:\myWebProject" (Windows) or "~/myWebProject" (Mac OS), as "JdbcUpdateTest.java".

```
1 import java.sql.*; // Use classes in java.sql package
2 
3 // JDK 1.7 and above
```

```
public class JdbcUpdateTest { // Save as "JdbcUpdateTest.java"
4
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
           ) {
              // Step 3 & 4: Execute a SQL UPDATE via executeUpdate()
14
              // which returns an int indicating the number of rows affected.
15
              // Increase the price by 7% and qty by 1 for id=1001
16
              String strUpdate = "update books set price = price*0.7, qty = qty+1 where id = 1001";
17
              System.out.println("The SQL query is: " + strUpdate); // Echo for debugging
18
19
              int countUpdated = stmt.executeUpdate(strUpdate);
20
              System.out.println(countUpdated + " records affected.");
21
22
              // Step 3 & 4: Issue a SELECT to check the UPDATE.
23
              String strSelect = "select * from books where id = 1001";
              System.out.println("The SQL query is: " + strSelect); // Echo for debugging
24
25
              ResultSet rset = stmt.executeQuery(strSelect);
26
              while(rset.next()) { // Move the cursor to the next row
                 System.out.println(rset.getInt("id") + ", "
27
                         + rset.getString("author") + ",
28
                         + rset.getString("title") + ",
29
                         + rset.getDouble("price") + ", "
30
31
                         + rset.getInt("qty"));
32
              }
33
           } catch(SQLException ex) {
34
              ex.printStackTrace();
35
           }
36
           // Step 5: Close the resources - Done automatically by try-with-resources
37
        }
38
    }
```

See the previous example on how to compile and run this program! Point to this Java file "JdbcUpdateTest"!

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".

16

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.

Save the program into your project directory "c:\myWebProject" (Windows) or "~/myWebProject" (Mac OS), as "JdbcInsertTest.java".

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

```
17
              // DELETE records with id>=3000 and id<4000
18
              String sqlDelete = "delete from books where id>=3000 and id<4000";</pre>
19
              System.out.println("The SQL query is: " + sqlDelete); // Echo for debugging
20
              int countDeleted = stmt.executeUpdate(sqlDelete);
21
              System.out.println(countDeleted + " records deleted.\n");
22
23
              // INSERT a record
              String sqlInsert = "insert into books " // need a space
24
              + "values (3001, 'Gone Fishing', 'Kumar', 11.11, 11)";
System.out.println("The SQL query is: " + sqlInsert); // Echo for debugging
25
26
27
              int countInserted = stmt.executeUpdate(sqlInsert);
              System.out.println(countInserted + " records inserted.\n");
28
29
30
              // INSERT multiple records
31
              sqlInsert = "insert into books values "
32
                    + "(3002, 'Gone Fishing 2', 'Kumar', 22.22, 22),"
                     + "(3003, 'Gone Fishing 3', 'Kumar', 33.33, 33)";
33
              System.out.println("The SQL query is: " + sqlInsert); // Echo for debugging
34
35
              countInserted = stmt.executeUpdate(sqlInsert);
              System.out.println(countInserted + " records inserted.\n");
36
37
              // INSERT a partial record
38
39
              sqlInsert = "insert into books (id, title, author) "
40
                    + "values (3004, 'Fishing 101', 'Kumar')";
              System.out.println("The SQL query is: " + sqlInsert); // Echo for debugging
41
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") + ",
                          + rset.getString("title") + ", "
52
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
     }
```

See the previous example on how to compile and run this program! Point to this Java file "JdbcInsertTest"!

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://www.oracle.com/technetwork/java/javase/jdbc/index.html.
- 3. White Fisher, et al., "JDBC API Tutorial and Reference", 3rd eds, Addison Wesley, 2003.
- 4. MySQL Home Page @ https://dev.mysql.com/, and documentation.
- 5. MySQL 5.7 Reference Manual @ https://dev.mysql.com/doc/refman/5.7en/.
- 6. Russell Dyer, "MySQL in a Nutshell", O'Reilly, 2008.

Latest version tested: Latest version tested: JDK 9.0.4, MySQL 5.7.21, Connector/J 5.1.45, Windows 10, Mac OS 10.13 Last modified: February, 2018

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