MIS 768: Advanced Software Concepts

Spring 2024

Database Applications (1)

Purpose

- Make connections from a Java program to database
- Submit SQL statements to database
- Handle the result of executing SQL statements

1. Preparation

(1) In this lab, we will use **MySQL** as our database environment for our Java programs. You can install the **MySQL server** by following the installation instructions (**17_installing_MySQL (Mac users)** or **17_installing_MySQL (Windows users))**.

(2) please make sure the **MySQL** server is up and running and you have the username and password ready.

(3) Launch Eclipse. Create a new package to hold our source file. Name the package as **edu.unlv.mis768.labwork17.**

(4) Download **17_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into the package.

2. Create the database in MySQL

(5) Open **CreateDB.java**.

(6) This program creates the database in MySQL for the lab examples. Please change the username and password you set up when you install your MySQL server.

```
6  public class CreateDB {
7          public static void main(String[] args) {
8             // Create a named constant for the URL.
9             // NOTE: This value is specific for MySQL.
10            final String DB_URL = "jdbc:mysql://localhost:3306/";
11            final String DB_COFFEE_URL = "jdbc:mysql://localhost:3306/coffeeData";
12            final String USERNAME = "root";
13            final String PASSWORD = "%^%$^&@4346_";
14
```

(7) Once you see the following messages in the Console, you are good to go.

```
Checking for existing database.
Database coffeeShopData created.
Coffee table created.
Customer table created.
UnpaidOrder table created.
```

3. Database Connection

(8) Create a new class named **TestConnection.java**. Enter the following code to test the database connection commands.

Please do not forget to replace lines 10-11 with your own username and password.

```
2
3  import java.sql.*;
4
5  public class TestConnection {
6
7      public static void main(String[] args) {
8          // constants for database connections
9          final String DB_URL = "jdbc:mysql://localhost:3306/coffeeShopData";
10         final String USERNAME = "root";
11         final String PASSWORD = "";
12
13         try {
14             // Create a connection to the database
15             Connection conn =  DriverManager.getConnection(DB_URL,USERNAME, PASSWORD);
16             // a prompt showing the connection is successful
17             System.out.println("Connetion to coffee DB is created.");
18
19
20             // close connection
21             conn.close();
22             System.out.println("connection closed.");
23
24         } catch (SQLException e) {
25             System.out.println("ERROR: "+e.getMessage());
26         }
27     }
28
29 }
```

4. Query Data

(9) We will add some statements in **TestConnection.java** to execute SQL statements.

```
13         try {
14             // Create a connection to the database
15             Connection conn =  DriverManager.getConnection(DB_URL,USERNAME, PASSWORD);
16             // a prompt showing the connection is successful
17             System.out.println("Connetion to coffee DB is created.");
18
19             // get the statement object
20             Statement stmt = conn.createStatement();
21             // prepare the SQL command
22             String sql = "SELECT Description, ProdNum, Price FROM Coffee";
23             // execute the query and get the result
24             ResultSet result = stmt.executeQuery(sql);
25
26             while(result.next()) {
27                 System.out.println("Description:"+ result.getString("Description")
28                 +"\tPrice: "+ result.getDouble("Price"));
29             }
30             // close connection
31             conn.close();
32             System.out.println("connection closed.");
33
34         } catch (SQLException e) {
35             System.out.println("ERROR: "+e.getMessage());
36         }
```

5. Data Insertion

(10) Open **CoffeeInserter.java**.

The program would allow the user to input the description, product number, and price.

Then the program connects to the database, sets up the prepared statement object, and executes the SQL statement.

```
32          // The SQL statement to be executed
33          String sqlStatement = " Insert into Coffee (ProdNum, Description, Price) values (?,?,?)";
34
35          try {
36              // create a connection object to the database
37              Connection conn = DriverManager.getConnection(DB_URL, USER_NAME, PASSWORD);
38
39              // instantiate a PrepareStatement object using the SQL command
40              PreparedStatement prepStmt = conn.prepareStatement(sqlStatement);
41
42              // provide the values for insertion. The order should be the same as specified in the sqlSta
43              prepStmt.setString(1, prodNum);
44              prepStmt.setString(2, description);
45              prepStmt.setDouble(3, price);
46
47              // for debugging
48              System.out.println(prepStmt);
49
50              // execute the sql command, and get how many rows are affected.
51              int row = prepStmt.executeUpdate();
52
53              // show a confirmation mesage
54              System.out.print(row+" row has been inserted.");
55
56              // close the connection
57              conn.close();
58
59          } catch (SQLException e) {
60              System.out.println("ERROR: "+e.getMessage());
61          }
```

6. Query Data

(11) Please open **CustomerFinder.java**. In this program, the user will enter a customer number. We will then show the corresponding record in the database.

First, use a prepared statement for the query, and provide values for the prepared statement.

```
30          // Create a SELECT statement to get the specific row from the Customer table.
31          String sqlStatement = "SELECT Name, State, ZIP FROM Customer WHERE CustomerNumber = ?";
32
33          // instantiate a PrepareStatement object using the SQL command
34          PreparedStatement prepStmt = conn.prepareStatement(sqlStatement);
35
36          // provide the values for query.
37          prepStmt.setString(1, customerNum);
38
```

(12) If the record can be found, print the name, state, and the zip code. We use getString() method there because the three columns are defined as CHAR in the database.

```
39            // Send the SELECT statement to the DBMS.
40            ResultSet result = prepStmt.executeQuery();
41
42            // If the result is not empty (i.e., have data to be read)
43            if (result.next()) {
44                  // Display the customer.
45                  System.out.println("Name: " + result.getString("Name"));
46                  System.out.println("State: " + result.getString("State"));
47                  System.out.println("Zip: " + result.getString("Zip"));
48            }
49            else {
50                // show not found message
51                System.out.println("Customer not found");
52            }
53
```

(13) You can now run and test the program.

7. Update Records

(14) Please open **CoffeePriceUpdater.java**. The program finds a specific product and update its price. The program include two methods **findProduct**() and **updatePrice**().

It first calls **findProduct** () to determine whether the record exists. If yes, ask the user to input a new price, and then class **updatePrice**() to update the price.

(15) The **findProduct** () method will return a Boolean value. Please it as following.

```
58        public static boolean findProduct(Connection conn, String prodNum) throws SQLException {
59            boolean productFound; // Flag
60
61            // Create a SELECT statement to get the specified row from the Coffee table.
62            String sqlStatement = "SELECT * FROM Coffee WHERE ProdNum = ?";
63
64            // instantiate a PrepareStatement object using the SQL command
65            PreparedStatement prepStmt = conn.prepareStatement(sqlStatement);
66
67            // provide the values for query.
68            prepStmt.setString(1, prodNum);
69
70           // Send the SELECT statement to the DBMS.
71           ResultSet result = prepStmt.executeQuery();
72
73           // If the result is not empty (i.e., have data to be read)
74           if (result.next()) {
75                // Set the flag to indicate the product was found.
76                productFound = true;
77           }
78           else {
79              // Indicate the product was not found.
80              productFound = false;
81           }
82
83           return productFound;
84        }
```

(16) The **updatePrice()** method similarly will update the database using a prepared statement. The prepared statement contains to arguments.

Please complete it as following:

```
98      public static void updatePrice(Connection conn, String prodNum, double price) throws SQLException {
99          // Create an UPDATE statement to update the price for the specified product number.
100         String sqlStatement = "UPDATE Coffee SET Price = ? WHERE ProdNum = ?";
101
102         // instantiate a PrepareStatement object using the SQL command
103         PreparedStatement prepStmt = conn.prepareStatement(sqlStatement);
104
105         // provide the values for Update command.
106         prepStmt.setDouble(1, price);
107         prepStmt.setString(2, prodNum);
108
109         // Send the UPDATE statement to the DBMS.
110         int rows = prepStmt.executeUpdate();
111
112         // Display the results.
113         System.out.println(rows + " row(s) updated.");
114     }
```

(17) You can run and test the program now.

8. Delete Records

(18) Please open **CoffeeDeletion.java** to execute the deletion. This program is very similar to **CoffeePriceUpdater.java**. After calling **findProduct**() to determine whether the record exists, ask the user whether he/she wants to remove the record. If yes, call **deleteCoffee**() to remove the record.

(19) Please add a new method**.** Name it **deleteCoffee( ) .**

It accepts a **`Conn`** object for the database and the product number for the desired coffee as parameters.

Note: you can choose to do a try-catch statement, rather than throwing SQLExceptioin.

```
93      public static void deleteCoffee(Connection conn, String prodNum) throws SQLException {
94          // Create a DELTE statement to update the price for the specified product number.
95          String sqlStatement = "DELETE FROM Coffee WHERE ProdNum = ?";
96
97          // instantiate a PrepareStatement object using the SQL command
98          PreparedStatement prepStmt = conn.prepareStatement(sqlStatement);
99
100         // provide the values for Update command.
101         prepStmt.setString(1, prodNum);
102
103
104         // Send the UPDATE statement to the DBMS.
105         int rows = prepStmt.executeUpdate();
106
107         // Display the results.
108         System.out.println(rows + " row(s) deleted.");
109
110     }
```

(20) We can then call the **deleteCoffee**() method in the **if** block as the following.

```
28              // Display the coffee's current data.
29              if (findProduct(conn, prodNum)) {
30                 // Make sure the user wants to delete this product.
31                 System.out.print("Are you sure you want to delete this item? (y/n): ");
32                 String sure = keyboard.nextLine();
33
34                 if (Character.toUpperCase(sure.charAt(0)) == 'Y') {
35                    // Delete the specified coffee.
36                    deleteCoffee(conn, prodNum);
37                 }
38                 else {
39                    System.out.println("The item was not deleted.");
40                 }
41
42               }
43               else  {
44                 // The specified product number was not found.
45                 System.out.println("That product was not found.");
46               }
47
```

(21) You can use product number 14-001 to test the program.