CE 223 DATABASE SYSTEMS

SECTION 3 & 4 - LAB 5

TASK 1: PERFORM A DATABASE QUERY AND VIEW RESULTS

You use a Statement object to perform database commands. To issue an SQL SELECT, we'll use the `statement.executeQuery(string)` method.

We will query the **`city`** table for all cities named "Hamburg".

```
// compose the SQL we want to use
String query = "SELECT * FROM city WHERE name='Hamburg'";
ResultSet rs = statement.executeQuery( query );
// The ResultSet is never null. The next() method iterates over results.
while (rs.next()) {
    String name = rs.getString("name");
    String district = rs.getString("district");
    String country = rs.getString("countrycode");
    int population = rs.getInt("population");
    System.out.printf("%s, %s, %s   pop. %d\n",
                      name, district, country, population);
}
```

USAGE OF RESULTSET

ResultSet is a "live" connection to rows in a database table. You can use ResultSet to read, test, and *modify* contents of a database. ResultSet methods that "get" data have 2 forms:

1) Get data by field number (first field in number 1, not 0):

```
String name = rs.getString( 2 );  // get 2nd field as a string
```

2) Get data by field name:

```
int population = rs.getInt( "population" );  // get field by name
```

ResultSet also has methods to test or change the current position in the results.

- **QUESTION 1: (25 PTS)**

1. Add the above code to your application and run it.

2. Modify this code to *ask the user for a city name* instead of "Hamburg". A city name may contain spaces, so you need to read an *entire input line* as city name.

3. Create a loop and ask city names until the user enters **"STOP"**.

4. Write your modified code below.

```
import java.sql.*;
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      try {
         Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/world",
"username", "password");
         Statement statement = connection.createStatement();
         String cityName;

         do {
            cityName = getCityName(scanner);
            if (!cityName.equalsIgnoreCase("STOP")) {
               String query = "SELECT * FROM city WHERE name='" + cityName + "';";
               ResultSet rs = statement.executeQuery(query);

               while (rs.next()) {
                  String name = rs.getString("name");
                  String district = rs.getString("district");
                  String country = rs.getString("countrycode");
                  int population = rs.getInt("population");
                  System.out.printf("%s, %s, %s   pop. %d\n", name, district, country, population);
               }
            }
         } while (!cityName.equalsIgnoreCase("STOP"));

      } catch (SQLException e) {
         e.printStackTrace();
      } finally {
         scanner.close();
      }
   }

   private static String getCityName(Scanner scanner) {
      System.out.println("Please enter a city name or STOP to exit:");
      return scanner.nextLine();
   }
```

```
}
```

- **QUESTION 2: (25 PTS)**

1. Find those countries and their populations where official language is French and the population is greater than 2 million.

2. Write the query with the Java code and the results below.

CODE:

```
import java.sql.*;

public class App {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/world";
        String username = "root";
        String password = "Passw0rd!";
        String query = "SELECT c.Name, c.Population "
            + "FROM country c "
            + "INNER JOIN CountryLanguage cl ON c.Code = cl.CountryCode "
            + "WHERE c.population > 2000000 AND cl.Language = 'French' AND
cl.IsOfficial = 'T'";

        try (Connection con = DriverManager.getConnection(url, username,
password);
             Statement statement = con.createStatement();
             ResultSet rs = statement.executeQuery(query)) {

            while (rs.next()) {
                String country = rs.getString("Name");
                double population = rs.getDouble("Population");
                System.out.printf("%s:    Population:    %.2f\n",    country,
population);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

RESULT SET:

BURUNDI, Population: 6695000
BELGIUM, Population: 10239000
CANADA, Population: 31147000
SWITZERLAND, Population: 7160400
FRANCE, Population: 59225700
HAITI, Population: 8222000
MADAGASCAR, Population: 15942000
RWANDA, Population: 7733000

- **QUESTION 3: (25 PTS)**

Read the Java API doc for ResultSet and **write** the method names in this table:

Method Name	Description
`isBeforeFirst()`	*Test* if the current position is *before* the first row in ResultSet. Returns false if the ResultSet is empty
`First()`	*Move* the current position to the first row of data in ResultSet. Returns true if successful. Returns false if there are no results.
`Next()`	*Test* if there are more results in ResultSet. If true, move current position to the next result.
`isLast()`	*Test* if the current position is the last result in ResultSet.

`Close()`	*Close* the ResultSet and release all its resources.

P.S.: ResultSet doesn't have a "hasNext" method.

TASK 2: USING A PREPAREDSTATEMENT

A *prepared statement* is an SQL command that is pre-compiled rather than interpreting the SQL during execution. A prepared statement can contain *placeholders* (?) where you insert values before executing the statement. Use Connection to create a Prepared Statement.

For example, to find all cities having a given `CountryCode`:

```
PreparedStatement pstmt = connection.prepareStatement(
                              "SELECT * FROM city WHERE countrycode=?" );
// insert a value for countrycode into PreparedStatement
pstmt.setString( 1, "THA" ); // replace the 1st ? with string "THA"
ResultSet rs = pstmt.executeQuery( );
```

We specify the SQL SELECT query when we ***create*** the prepared statement. The ? in the query is a *placeholder* where you can insert a value later. Do not put quotes around ?, even if the value will be a string. The PreparedStatement will take care of that.

The first ? in a prepared statement is parameter 1, the second ? is parameter 2, etc.

- **QUESTION 4: (25 PTS)**

Add a **new city** to the database, with the name as **Central City** that belongs to the **United States of America**, the district as **Central District**, and the population as **50010** by using PreparedStatements.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class AddCityToDatabase {
    public static void main(String[] args) {
        // Database connection details
        String url = "jdbc:mysql://localhost:3306/your_database_name";
        String username = "your_username";
        String password = "your_password";

        // City details
        String cityName = "Central City";
        String country = "United States of America";
        String district = "Central District";
        int population = 50010;

        // SQL query
        String sql = String.format(
            "INSERT INTO cities (name, country, district, population) VALUES ('%s', '%s', '%s', %d)",
            cityName, country, district, population);

        Connection conn = null;
        Statement stmt = null;
```

```
        try {
            // Establish a connection
            conn = DriverManager.getConnection(url, username, password);
            stmt = conn.createStatement();

            // Execute the query
            stmt.executeUpdate(sql);

            System.out.println("City added successfully.");

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // Close connections
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e) { /* ignored */ }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) { /* ignored */ }
            }
        }
    }
}
```

26.05.2023