

Name: Vaishali Ramesh Kale

Email id : kalevaishalir16@gmail.com

Day 21:

Task 1: Establishing Database Connections Write a Java program that connects to a SQLite database and prints out the connection object to confirm successful connection.

Solution::::

Explanation

1. Load the JDBC Driver:
 - `Class.forName("org.sqlite.JDBC");` is used to load the SQLite JDBC driver. This step is optional for JDBC 4.0 and later as the driver manager can automatically load the driver.
2. Establish a Connection:
 - `String url = "jdbc:sqlite:sample.db";` specifies the database file. If the file does not exist, SQLite will create it.
 - `connection = DriverManager.getConnection(url);` establishes the connection to the database.
3. Print the Connection Object:
 - `System.out.println(connection);` prints the connection object to confirm a successful connection.
4. Exception Handling:
 - Catch and handle `SQLException` and `ClassNotFoundException` to deal with any issues in connecting to the database or loading the driver.
5. Close the Connection:
 - Always close the connection in a finally block to ensure it's closed even if an exception occurs.
 - Running the Program
 - Ensure you have the SQLite JDBC driver in your project's classpath.
 - Compile and run the Java program.
 - You should see output confirming the successful connection and the connection object.

CODE:::

```
package com.wipro.util;

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

public class DBConnection {

    public static Connection con;

    public static Connection getMyDBConn()
    {

        try {

            con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc1", "root",
"Pass@1234");

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return con;

    }

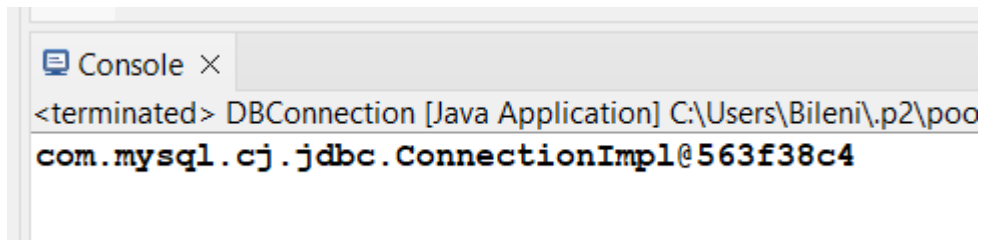
    public static void main(String[] args) {

System.out.println(getMyDBConn());

    }

}
```

OUTPUT:::



```
Console ×
<terminated> DBConnection [Java Application] C:\Users\Bileni\.p2\poo
com.mysql.cj.jdbc.ConnectionImpl@563f38c4
```

Task 2: SQL Queries using JDBC Create a table 'User' with a following schema 'User ID' and 'Password' stored as hash format (note you have research on how to generate hash from a string), accept "User ID" and "Password" as input and check in the table if they match to confirm whether user access is allowed or not.

Solution:::

Explanation:

1. Insert User:

- The insertUser method hashes the password and inserts the user ID and hashed password into the User table.

2. Authenticate User:

- The authenticateUser method retrieves the hashed password from the database and compares it with the hashed password provided by the user.

3. Debugging:

- Print statements are added to display the hashed password being inserted and the hashed password being checked.

• **Database Connection:**

The DB_URL, DB_USER, and DB_PASSWORD variables hold the database connection details.

• **User Input:**

The program prompts the user to enter a User ID and Password.

• **Password Hashing:**

The PasswordUtils.hashPassword method is used to hash the entered password.

• **User Authentication:**

The authenticateUser method checks if the provided User ID and hashed password match any entry in the User table.

If a match is found, access is granted; otherwise, it is denied.

CODE:::

Database Query:

```
1
2 • CREATE DATABASE userdb;
3 • USE userdb;
4 • show tables;
5 • CREATE TABLE User (
6     user_id VARCHAR(50) PRIMARY KEY,
7     password_hash VARCHAR(64) NOT NULL
8 );
9 • create user 'kale'@'localhost' identified by 'root';
10 • grant all privileges on userdb.* to 'kale'@'localhost';
11
```

Java Code:::

```
package com.wipro.advancejdbc;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class PasswordUtils {
    public static String hashPassword(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(password.getBytes());
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package com.wipro.advancejdbc;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```

import java.sql.SQLException;
import java.util.Scanner;

public class UserAuthentication {

    private static final String DB_URL = "jdbc:mysql://localhost:3306/userdb";
    private static final String DB_USER = "kale"; // Replace with your DB username
    private static final String DB_PASSWORD = "root"; // Replace with your DB password

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

        // Inserting a new user (run this once)
        insertUser("kale", "root");

        System.out.println("Enter User ID:");
        String userId = scanner.nextLine();

        System.out.println("Enter Password:");
        String password = scanner.nextLine();

        // Hash the password
        String passwordHash = PasswordUtils.hashPassword(password);
        System.out.println("Entered password hash: " + passwordHash);

        // Check user credentials
        if (authenticateUser(userId, passwordHash)) {
            System.out.println("Access granted!");
        } else {
            System.out.println("Access denied!");
        }

        scanner.close();
    }

    public static void insertUser(String userId, String password) {
        try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD)) {
            String passwordHash = PasswordUtils.hashPassword(password);
            String query = "INSERT INTO User (user_id, password_hash) VALUES (?, ?)";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, userId);
            preparedStatement.setString(2, passwordHash);
            preparedStatement.executeUpdate();
            System.out.println("User inserted with hash: " + passwordHash);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

public static boolean authenticateUser(String userId, String passwordHash) {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD)) {
        String query = "SELECT * FROM User WHERE user_id = ? AND password_hash = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, userId);
        preparedStatement.setString(2, passwordHash);

        ResultSet resultSet = preparedStatement.executeQuery();
        return resultSet.next(); // Returns true if a match is found
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
}
}

```

OUTPUT:::



```

// Check user credentials

<terminated> UserAuthentication (1) [Java Application] C:\Users\Bilen\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1
User inserted with hash: 4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Enter User ID:
kale
Enter Password:
root
Entered password hash: 4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Access granted!

```

Task 3: PreparedStatement Modify the SELECT query program to use PreparedStatement to parameterize the query and prevent SQL injection.

Solution:::

Explanation:

1. PreparedStatement Usage:

- The authenticateUser method now uses PreparedStatement to create the SQL query. This ensures that the user inputs (userId and passwordHash) are treated as parameters and not as part of the SQL query, preventing SQL injection.

2. Setting Parameters:

- The setString method is used to set the userId and passwordHash values in the query.

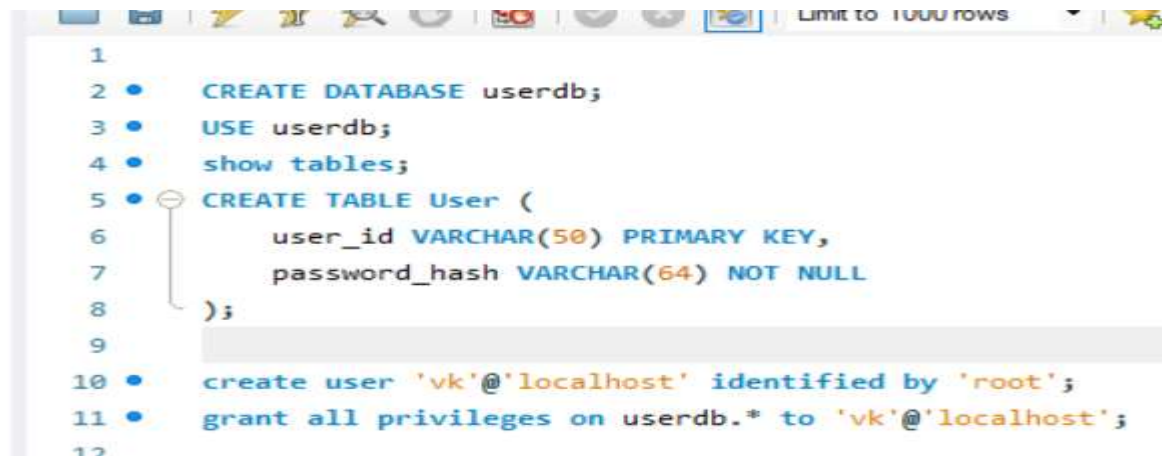
3. Execute Query:

- The executeQuery method of PreparedStatement is used to execute the query.

4. Resource Management:

- The try-with-resources statement is used to ensure that the Connection and PreparedStatement are closed automatically, even if an exception occurs.

Database query::



```
1
2 • CREATE DATABASE userdb;
3 • USE userdb;
4 • show tables;
5 • CREATE TABLE User (
6     user_id VARCHAR(50) PRIMARY KEY,
7     password_hash VARCHAR(64) NOT NULL
8 );
9
10 • create user 'vk'@'localhost' identified by 'root';
11 • grant all privileges on userdb.* to 'vk'@'localhost';
12
```

Java CODE:::

```
package com.wipro.preparedstmt;
```

```
import java.security.MessageDigest;
```

```
import java.security.NoSuchAlgorithmException;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
```

```
import java.sql.SQLException;

import java.util.Scanner;

public class UserAuthentication {

    private static final String DB_URL = "jdbc:mysql://localhost:3306/userdb";
    private static final String DB_USER = "vk"; // Replace with your DB username
    private static final String DB_PASSWORD = "root"; // Replace with your DB password

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Inserting a new user (run this once)
        insertUser("vk", "root");

        System.out.println("Enter User ID:");
        String userId = scanner.nextLine();

        System.out.println("Enter Password:");
        String password = scanner.nextLine();

        // Hash the password
        String passwordHash = PasswordUtils.hashPassword(password);
        System.out.println("Entered password hash: " + passwordHash);

        // Check user credentials
        if (authenticateUser(userId, passwordHash)) {
            System.out.println("Access granted!");
        } else {
```



```

        System.out.println("Access denied!");
    }

    scanner.close();
}

public static void insertUser(String userId, String password) {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD)) {
        String passwordHash = PasswordUtils.hashPassword(password);
        String query = "INSERT INTO User (user_id, password_hash) VALUES (?, ?)";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, userId);
        preparedStatement.setString(2, passwordHash);
        preparedStatement.executeUpdate();
        System.out.println("User inserted with hash: " + passwordHash);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static boolean authenticateUser(String userId, String passwordHash) {
    String query = "SELECT * FROM User WHERE user_id = ? AND password_hash = ?";
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
        PreparedStatement preparedStatement = connection.prepareStatement(query)) {
        preparedStatement.setString(1, userId);
        preparedStatement.setString(2, passwordHash);

        ResultSet resultSet = preparedStatement.executeQuery();
    }
}

```

```

        return resultSet.next(); // Returns true if a match is found
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
}
}

```

```

class PasswordUtils {
    public static String hashPassword(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(password.getBytes());
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}
}

```

OUTPUT:::

By entering Correct credentials:

```
Console X
<terminated> UserAuthentication (2) [Java Application] C:\Users\Bileni\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v2
User inserted with hash: 4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Enter User ID:
vk
Enter Password:
root
Entered password hash: 4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Access granted!
```