prajwalhonashetti143@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.

To create a Java program that connects to a SQLite database and prints out the connection object to confirm a successful connection, follow these steps:

- Set up your project: Ensure you have the SQLite JDBC library. If you are using an IDE like IntelliJ IDEA or Eclipse, you can add the JDBC library to your project. If you are using Maven, you can include the dependency in your pom.xml.
- Write the Java code: The code will establish a connection to the SQLite database and print the connection object.

Step 1: Add SQLite JDBC library

If you are using Maven, add the following dependency to your pom.xml:

```
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.41.2.1</version> <!-- Check for the latest version -->
</dependency>
```

Step 2: Write the Java code:

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

public class SQLiteConnectionTest {

    public static void main(String[] args) {
        Connection connection = null;
```

```
        try {
DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());
connection    =DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb","root",
"admin");  //write your own database details


            System.out.println("Connection to SQLite has been established.");
            System.out.println("Connection object: " + connection);


        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

Explanation:

DriverManager.getConnection(url): Establishes a connection to the SQLite database specified by the URL.

Print statements: Confirm that the connection is established by printing a success message and the connection object.

Exception handling: Catches any SQLException that might occur during the connection process and prints the error message.

Connection close in finally block: Ensures that the connection is closed properly after the operation is completed.

Running the program

- **Connection to SQLite has been established.**
- **Connection object: org.sqlite.jdbc4.JDBC4Connection@xxxxxxxx**

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.

To create a Java program that performs the following tasks:

- Connects to a SQLite database.
- Creates a table named User with columns UserID and Password (stored as a hash).
- Accepts UserID and Password as input.
- Hashes the password.
- Checks if the input UserID and hashed password match a record in the table to confirm user access.

```
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.sql.Statement;

import java.util.Scanner;


public class UserAccessControl {


   private static final String DATABASE_URL = "jdbc:sqlite:userdb.db"; //own database details
```

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

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

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

    String hashedPassword = hashPassword(password);

    if (checkUserAccess(userId, hashedPassword)) {
        System.out.println("Access Granted.");
    } else {
        System.out.println("Access Denied.");
    }

    scanner.close();
}

private static void createTable() {
    String createTableSQL = "CREATE TABLE IF NOT EXISTS User (" +
                            "UserID TEXT PRIMARY KEY, " +
                            "Password TEXT NOT NULL)";

    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
```

```
        Statement stmt = conn.createStatement()) {
        stmt.execute(createTableSQL);
        System.out.println("User table created or already exists.");
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

private static void insertSampleUser() {
    String userId = "alice";
    String password = "secret123";
    String hashedPassword = hashPassword(password);

    String insertSQL = "INSERT OR IGNORE INTO User (UserID, Password) VALUES (?, ?)";

    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
         PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {
        pstmt.setString(1, userId);
        pstmt.setString(2, hashedPassword);
        pstmt.executeUpdate();
        System.out.println("Sample user inserted or already exists.");
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

private static String hashPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
```

```
        byte[] hash = md.digest(password.getBytes());
        StringBuilder hexString = new StringBuilder(2 * hash.length);
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}

private static boolean checkUserAccess(String userId, String hashedPassword) {
    String query = "SELECT COUNT(1) FROM User WHERE UserID = ? AND Password = ?";

    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
         PreparedStatement pstmt = conn.prepareStatement(query)) {
        pstmt.setString(1, userId);
        pstmt.setString(2, hashedPassword);

        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt(1) > 0;
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
```

```
        }

        return false;
    }
}
```

Running the Program:

Enter User ID: alice

Enter Password: secret123

Expected output:

Enter User ID: alice

Enter Password: secret123

Access Granted.

Task 3: PreparedStatement

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

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

public class UserAccessControl {
```

```
    private static final String DATABASE_URL = "jdbc:sqlite:userdb.db"; // write your database path

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

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

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

        String hashedPassword = hashPassword(password);

        if (checkUserAccess(userId, hashedPassword)) {
            System.out.println("Access Granted.");
        } else {
            System.out.println("Access Denied.");
        }

        scanner.close();
    }

    private static void createTable() {
        String createTableSQL = "CREATE TABLE IF NOT EXISTS User (" +
                                "UserID TEXT PRIMARY KEY, " +
                                "Password TEXT NOT NULL)";
```

```
    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
         Statement stmt = conn.createStatement()) {
        stmt.execute(createTableSQL);
        System.out.println("User table created or already exists.");
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

private static void insertSampleUser() {
    String userId = "alice";
    String password = "secret123";
    String hashedPassword = hashPassword(password);

    String insertSQL = "INSERT OR IGNORE INTO User (UserID, Password) VALUES (?, ?)";

    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
         PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {
        pstmt.setString(1, userId);
        pstmt.setString(2, hashedPassword);
        pstmt.executeUpdate();
        System.out.println("Sample user inserted or already exists.");
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

private static String hashPassword(String password) {
    try {
```

```
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] hash = md.digest(password.getBytes());
        StringBuilder hexString = new StringBuilder(2 * hash.length);
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}

private static boolean checkUserAccess(String userId, String hashedPassword) {
    String query = "SELECT COUNT(1) FROM User WHERE UserID = ? AND Password = ?";

    try (Connection conn = DriverManager.getConnection(DATABASE_URL);
         PreparedStatement pstmt = conn.prepareStatement(query)) {
        pstmt.setString(1, userId);
        pstmt.setString(2, hashedPassword);

        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt(1) > 0;
        }
    } catch (SQLException e) {
```

```
            System.out.println(e.getMessage());
        }

        return false;
    }
}
```

Explanation:

- **createTable() Method:** Creates the User table if it does not exist. This method uses a Statement because there are no user inputs involved, so SQL injection is not a concern here.
- **insertSampleUser() Method:** Inserts a sample user into the User table. The PreparedStatement is used to safely insert the user data.
- **hashPassword**() Method: Hashes a given password using SHA-256. This ensures that passwords are stored securely in the database.

checkUserAccess() Method:

- Uses a **PreparedStatement** to safely check if the provided **UserID** and hashed password match a record in the User table.
- The ? placeholders in the SQL query are replaced with the user-provided values using **pstmt.setString(1, userId) and pstmt.setString(2, hashedPassword).**
- This prevents SQL injection by ensuring that the input parameters are safely handled by the **PreparedStatement**.

Expected Output

User ID: alice

Enter Password: secret123

Access Granted.