Questions

1. Explain different steps involved in jdbc process with code snippets
2. Explain 4 Types of JDBC Drivers
3. List & Explain 3 types of Exceptions in JDBC
4. Write a program to connect
    1. (Driver:JDBC/ODBC Bridge,URL:'JDBC.ODBC:ex',Username: XYZ,Password :123)
    2. retrive all rows with marks > 60
    3. Assume the follwing table:
        1. Table name(Student)
        2. Fields:USN,Marks,Name
5. Explain different types of Statement objects with example
6. Discuss the scrollable result set and updateable result set
7. Write a java prog to execute a database transaction
8. Transaction Processing in jdbc

Answers

1. Steps Involved in JDBC Process with Code Snippets

1. Load the JDBC Driver:

```java
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

2. Establish a Connection:

```java
String url = "jdbc:mysql://localhost:3306/mydatabase";
String user = "username";
String password = "password";

Connection connection = null;
try {
    connection = DriverManager.getConnection(url, user, password);
} catch (SQLException e) {
    e.printStackTrace();
}
```

3. Create a Statement:

```java
Statement statement = null;
try {
    statement = connection.createStatement();
} catch (SQLException e) {
    e.printStackTrace();
}
```

4. Execute a Query:

```java
String sql = "SELECT * FROM mytable";
ResultSet resultSet = null;
try {
    resultSet = statement.executeQuery(sql);
} catch (SQLException e) {
    e.printStackTrace();
}
```

5. Process the ResultSet:

```java
try {
    while (resultSet.next()) {
        int id = resultSet.getInt("id");
        String name = resultSet.getString("name");
        System.out.println("ID: " + id + ", Name: " + name);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

6. Close the Connection:

```java
try {
    if (resultSet != null) resultSet.close();
    if (statement != null) statement.close();
    if (connection != null) connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

2. Types of JDBC Drivers

1. Type 1: JDBC-ODBC Bridge Driver:

- Translates JDBC calls into ODBC calls.
- Requires ODBC driver on the client machine.

- Example: `sun.jdbc.odbc.JdbcOdbcDriver`

2. Type 2: Native-API Driver:

- Converts JDBC calls into database-specific API calls.
- Requires database-specific native libraries.
- Example: `oracle.jdbc.driver.OracleDriver`

3. Type 3: Network Protocol Driver:

- Uses middleware to convert JDBC calls into database-specific calls.
- Example: `com.sybase.jdbc.SybDriver`

4. Type 4: Thin Driver:

- Directly converts JDBC calls into database-specific protocol calls.
- No native libraries required on client machine.
- Example: `com.mysql.cj.jdbc.Driver`

3. Types of Exceptions in JDBC

1. SQLException:

- General exception for database access errors.
- Example: `SQLException e`

2. SQLTimeoutException:

- Subclass of `SQLException`.
- Thrown when a timeout occurs.

3. SQLDataException:

- Subclass of `SQLException`.
- Thrown when data integrity issues occur.

4. Java Program for JDBC/ODBC Bridge

1. Load the JDBC Driver:

```
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

2. Connect to Database and Retrieve Rows:

```
import java.sql.*;
```

```
public class JDBCODBCExample {
    public static void main(String[] args) {
        String url = "jdbc:odbc:ex";
        String user = "XYZ";
        String password = "123";

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            connection = DriverManager.getConnection(url, user, password);
            statement = connection.createStatement();
            String sql = "SELECT * FROM Student WHERE Marks > 60";
            resultSet = statement.executeQuery(sql);

            while (resultSet.next()) {
                String usn = resultSet.getString("USN");
                int marks = resultSet.getInt("Marks");
                String name = resultSet.getString("Name");
                System.out.println("USN: " + usn + ", Marks: " + marks + ", Name:
" + name);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (resultSet != null) resultSet.close();
                if (statement != null) statement.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5. Types of Statement Objects

1. Statement:

- Used to execute simple SQL queries.

```
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("SELECT * FROM mytable");
```

2. PreparedStatement:

- Used for executing precompiled SQL queries with parameters.

```
PreparedStatement pstmt = connection.prepareStatement("SELECT * FROM mytable WHERE
id = ?");
pstmt.setInt(1, 1);
ResultSet rs = pstmt.executeQuery();
```

3. CallableStatement:

- Used to execute stored procedures.

```
CallableStatement cstmt = connection.prepareCall("{call myprocedure(?)}");
cstmt.setInt(1, 1);
ResultSet rs = cstmt.executeQuery();
```

6. Scrollable and Updatable Result Sets

Scrollable Result Set:

- Allows moving the cursor forward, backward, and to a specific row.

```
Statement stmt = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("SELECT * FROM mytable");
rs.last(); // Move to the last row
```

Updatable Result Set:

- Allows updating rows in the result set.

```
Statement stmt = connection.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM mytable");
if (rs.next()) {
    rs.updateString("name", "newname");
    rs.updateRow();
}
```

7. Program to Execute a Database Transaction

```
import java.sql.*;

public class TransactionExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "username";
        String password = "password";

        Connection connection = null;

        try {
            connection = DriverManager.getConnection(url, user, password);
            connection.setAutoCommit(false); // Start transaction

            Statement stmt = connection.createStatement();
            stmt.executeUpdate("UPDATE accounts SET balance = balance - 100 WHERE
account_id = 1");
            stmt.executeUpdate("UPDATE accounts SET balance = balance + 100 WHERE
account_id = 2");

            connection.commit(); // Commit transaction
        } catch (SQLException e) {
            try {
                if (connection != null) {
                    connection.rollback(); // Rollback transaction on error
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
        } finally {
            try {
                if (connection != null) connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

8. Transaction Processing in JDBC

Transaction processing involves managing a set of operations as a single unit of work. In JDBC, you can manage transactions using the following steps:

1. **Disable Auto-Commit Mode:**

```
connection.setAutoCommit(false);
```

2. **Execute SQL Statements:**

Execute the required SQL statements within the transaction.

3. **Commit the Transaction:**

```
connection.commit();
```

4. **Rollback the Transaction:**

If an error occurs, rollback the transaction to maintain data integrity.

```
connection.rollback();
```

Using these steps, you can ensure that a series of operations are executed as a single, atomic unit, preserving data consistency and integrity.