1. Introduction to JDBC

Theory Questions:

1. **What is JDBC (Java Database Connectivity)?**

⇒ JDBC (Java Database Connectivity) is an API (Application Programming Interface) provided by Java to connect and interact with databases. It allows Java applications to execute SQL statements such as **insert, update, delete, and query** to manage relational databases like MySQL, Oracle, PostgreSQL, etc. JDBC acts as a bridge between Java applications and the database, ensuring smooth communication and data handling.

2. **What is the importance of JDBC in Java programming?**

⇒ JDBC is important in Java programming for the following reasons:

- Database Connectivity: It provides a standard method to connect Java applications with various databases.
- Cross-Platform Support: JDBC works with different types of databases using different drivers.
- SQL Execution: It allows execution of SQL statements directly from Java code.
- Data Manipulation: JDBC enables reading, writing, updating, and deleting data from databases.
- Flexibility: It can be used with both standalone and enterprise applications.

3. **Explain JDBC architecture with the role of the following:**
    - **Driver Manager**
    - **Driver**
    - **Connection**
    - **Statement**
    - **ResultSet**

⇒ JDBC architecture consists of two main layers:

1. JDBC API – Interfaces like `Connection`, `Statement`, `ResultSet`, etc.

2. JDBC Driver – Translates Java API calls into database-specific calls.

Roles of components:

- DriverManager:
  Acts as a factory for creating connections. It manages all the database drivers and establishes a connection to the database by selecting the appropriate driver.

- Driver:
  A database-specific implementation that communicates with the database. It is registered with the `DriverManager`.

- Connection:
  Represents a session between the Java application and the database. It is used to send SQL statements to the database.

- Statement:
  Used to execute SQL queries (static SQL). It sends SQL commands through the Connection.

- ResultSet:
  A table of data returned by executing a SQL SELECT query. It allows traversal through the retrieved records.

Lab Questions:

4. Write a Java program to connect to a MySQL database using JDBC.

⇒ Here is a java program to connect to a MySQL DB using JDBC:

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

public class JDBCConnectionExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database_name";
        String user = "root";
        String password = "your_password";

        try {
            Connection conn = DriverManager.getConnection(url, user, password);
            System.out.println("Database connected successfully!");
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

5. Demonstrate how to load a JDBC driver and establish a connection.

⇒

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

public class LoadDriverExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database_name";
        String user = "root";
        String password = "your_password";

        try {
            // Step 1: Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish the connection
            Connection conn = DriverManager.getConnection(url, user, password);
            System.out.println("Connection established successfully!");

            // Step 3: Close the connection
            conn.close();
        } catch (ClassNotFoundException e) {
            System.out.println("JDBC Driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. JDBC Driver Types

Theory Questions:

6. What are the four types of JDBC drivers?

Type 1: JDBC-ODBC Bridge Driver

⇒Uses ODBC (Open Database Connectivity) to connect to the database.

- Example: `sun.jdbc.odbc.JdbcOdbcDriver`
- *Now mostly obsolete.*

Type 2: Native-API Driver

⇒ Converts JDBC calls into native database calls using native libraries (C/C++).

- Faster than Type 1 but platform-dependent.

Type 3: Network Protocol Driver

⇒ Converts JDBC calls into a database-independent network protocol, which is then translated to DB-specific protocol by a server.

- Useful in enterprise applications with middleware.

Type 4: Thin Driver

⇒Directly converts JDBC calls to the database-specific protocol using Java.

- No native code, platform-independent, and commonly used today (e.g., MySQL Connector/J).

7. **Compare the different types of JDBC drivers.**

⇒

Type	Description	Speed	Portability	Dependency
Type 1	**Uses ODBC bridge**	**Slowest**	**Platform dependent**	**Requires ODBC driver**
Type 2	**Uses native DB API**	**Fast**	**Platform dependent**	**Requires native library**

Type 3	**Uses middleware server**	**Moderate**	**Platform independent**	**Needs server-side component**
Type 4	**Pure Java driver**	**Fastest**	**Platform independent**	**No external dependency**

8. Which JDBC driver is most suitable for modern Java and MySQL environments?

⇒ The Type 4 driver (Thin Driver) is most suitable for modern Java and MySQL environments. It is:

- Pure Java-based
- Lightweight and fast
- Doesn't require any native libraries or middleware
- Example: MySQL Connector/J → `"com.mysql.cj.jdbc.Driver"`

Lab Questions:

9. Identify the JDBC driver used in your Java-MySQL program.

⇒ In Java-MySQL programs, the commonly used driver is:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

This is a Type 4 driver provided by MySQL Connector/J, which is widely used in modern applications for its performance and portability.

10. Research and explain the best JDBC driver for your system.

⇒ For systems using MySQL and Java, the best JDBC driver is:

✅ MySQL Connector/J

- Driver Class: `com.mysql.cj.jdbc.Driver`
- Driver Type: Type 4 (Pure Java)
- Why Best?
    - Easy to use with JDBC
    - Cross-platform
    - Compatible with modern MySQL versions
    - Actively maintained by Oracle

Make sure to download and add the `.jar` file of MySQL Connector/J in your project's classpath to use it.

---

3. Steps for Creating JDBC Connections

Theory Questions:

11. What are the steps to create a JDBC connection?

- **Import JDBC packages**

⇒ Include required classes from `java.sql` package.

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

- **Register JDBC driver**

⇒ Load the database driver class using `Class.forName()`.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- **Open database connection**

⇒ Use `DriverManager.getConnection()` to establish a connection.

```
Connection conn = DriverManager.getConnection(url, user,
password);
```

- **Create a statement**

⇒ Use `conn.createStatement()` or `conn.prepareStatement()` to create an SQL statement.

```
Statement stmt = conn.createStatement();
```

- **Execute SQL queries**

⇒ Execute query using `executeQuery()` or `executeUpdate()` depending on SQL type.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```

- **Process the ResultSet**

⇒ Loop through the results and access data using methods like `getString()`, `getInt()`.

```
while(rs.next()) {
    System.out.println(rs.getString("name"));
}
```

- **Close the connection**

⇒ Properly close `ResultSet`, `Statement`, and `Connection` to release resources.

```
rs.close();

stmt.close();

conn.close();
```

Lab Questions:

12. Write a Java program to establish a JDBC connection and print a success message.

⇒

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

public class JDBCConnect {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database_name";
        String user = "root";
        String password = "your_password";

        try {
            // Load JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establish connection
            Connection conn = DriverManager.getConnection(url, user, password);

            // Success message
            System.out.println("✅ JDBC Connection established successfully!");

            // Close connection
            conn.close();
        } catch (ClassNotFoundException e) {
            System.out.println("JDBC Driver not found!");
            e.printStackTrace();
        } catch (SQLException e) {
            System.out.println("Connection failed!");
            e.printStackTrace();
        }
    }
}
```

---

4. Types of JDBC Statements

Theory Questions:

13. What is a Statement in JDBC?

⇒ A Statement in JDBC is used to execute static SQL queries against the database. It is created using the `Connection` object and is suitable when the SQL statement doesn't require parameters.

Created using:

```
Statement stmt = conn.createStatement();
```

Used with:

- `executeQuery()` – for SELECT queries
- `executeUpdate()` – for INSERT, UPDATE, DELETE

Not recommended when dealing with user inputs as it is vulnerable to SQL injection.

14. What is a PreparedStatement in JDBC?

⇒ A PreparedStatement is a precompiled SQL statement used for executing parameterized queries. It is more secure and efficient than a Statement, especially when the same query is executed multiple times with different values.

Created using:

```
PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM
users WHERE id = ?");
```

Set parameters:

```
pstmt.setInt(1, 5);
```

Benefits:

- Protects against SQL injection
- Increases performance through query reuse

15. What is a CallableStatement in JDBC?

⇒ A CallableStatement is used to execute stored procedures in the database. It can handle IN, OUT, and INOUT parameters and allows complex logic to be executed within the database.

🔹 Created using:

```
CallableStatement cstmt = conn.prepareCall("{call
procedure_name(?, ?)}");
```

🔹 Use `.setXXX()` and `.registerOutParameter()` methods to work with IN/OUT parameters.

16. Compare Statement, PreparedStatement, and CallableStatement.

⇒

Feature	Statement	PreparedStatement	CallableStatement
Purpose	**Execute static SQL**	**Execute parameterized queries**	**Execute stored procedures**
Security	**Low (SQL Injection risk)**	**High (Prevents SQL Injection)**	**High (Executes procedures securely)**

Performance	**Low**	**precompiled**	**Depends on stored procedure complexity**
Reusability	**No**	**Yes**	**Yes**
Parameters	**Not Supported**	**Uses ? placeholders**	**Supports `IN`, `OUT`, `INOUT`**

Lab Questions:

17. Create a program using Statement to perform CRUD operations.

⇒

```
import java.sql.*;


public class CRUDUsingStatement {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/Demo";

        String user = "root";

        String password = "";


        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            Connection conn = DriverManager.getConnection(url,
user, password);

            Statement stmt = conn.createStatement();
```

```
            stmt.executeUpdate("INSERT INTO student (id, name)
        VALUES (1, 'Ayush')");


            stmt.executeUpdate("UPDATE student SET name = 'Ayush
Patel' WHERE id = 1");


            ResultSet rs = stmt.executeQuery("SELECT * FROM
student");
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("id") + ",
Name: " + rs.getString("name"));
            }


            // DELETE
            stmt.executeUpdate("DELETE FROM student WHERE id =
1");
            rs.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
}
```

18. Modify the program to use PreparedStatement for parameterized queries.

⇒

```
import java.sql.*;

public class CRUDUsingPreparedStatement {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/Demo";

        String user = "root";

        String password = "";

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            Connection conn = DriverManager.getConnection(url,
user, password);

            PreparedStatement insertStmt =
conn.prepareStatement("INSERT INTO student (id, name) VALUES (?,
?)");

            insertStmt.setInt(1, 1);

            insertStmt.setString(2, "Ayush");

            insertStmt.executeUpdate();

            PreparedStatement updateStmt =
conn.prepareStatement("UPDATE student SET name = ? WHERE id =
?");
```

```
            updateStmt.setString(1, "Ayush Patel");

            updateStmt.setInt(2, 1);

            updateStmt.executeUpdate();

            PreparedStatement selectStmt =
conn.prepareStatement("SELECT * FROM student");

            ResultSet rs = selectStmt.executeQuery();

            while (rs.next()) {

                System.out.println("ID: " + rs.getInt("id") + ",
Name: " + rs.getString("name"));

            }

            PreparedStatement deleteStmt =
conn.prepareStatement("DELETE FROM student WHERE id = ?");

            deleteStmt.setInt(1, 1);

            deleteStmt.executeUpdate();



            rs.close();

            insertStmt.close();

            updateStmt.close();

            selectStmt.close();

            deleteStmt.close();

            conn.close();

        } catch (Exception e) {

            e.printStackTrace();
```

```
        }
    }
}
```

---

5. JDBC CRUD Operations (Insert, Update, Select, Delete)

LAB :

Insert

```
String sql = "INSERT INTO students (name, age) VALUES (?, ?)";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, "Ayush");
stmt.setInt(2, 21);
stmt.executeUpdate();
```

Update

```
String sql = "SELECT * FROM students";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
    System.out.println(rs.getString("name") + " - " + rs.getInt("age"));
}
```

Select/display

```
String sql = "SELECT * FROM students";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
    System.out.println(rs.getString("name") + " - " + rs.getInt("age"));
}
```

Delete

```
String sql = "DELETE FROM students WHERE name = ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, "Ayush");
stmt.executeUpdate();
```

Theory questions

o Insert: Adding a new record to the database.

⇒ In JDBC, the `INSERT` statement is used to add new records into a database table.

It is commonly executed using `PreparedStatement` for parameterized and secure queries.

The `executeUpdate()` method is called to run the query and insert the data.

o Update: Modifying existing records.

⇒ In JDBC, the `UPDATE` statement is used to modify existing records in a table.

It is often executed with `PreparedStatement` to update specific columns based on conditions.

The `executeUpdate()` method is used to apply the changes to the database.

o Select: Retrieving recordsfrom the database.

⇒ In JDBC, the `SELECT` statement retrieves records from a database table.

It is executed using `Statement` or `PreparedStatement`, and results are accessed via a `ResultSet` object.

The `executeQuery()` method runs the query and returns the data for processing.

o Delete: Removing records from the database.

⇒ In JDBC, the `DELETE` statement removes records from a database table.
It is usually executed with `PreparedStatement` to delete specific rows based on conditions.
The `executeUpdate()` method is used to perform the deletion in the database.

---

6. ResultSet Interface

Theory Questions:

24. What is a ResultSet in JDBC?

⇒ A `ResultSet` in JDBC is an object that holds the data returned by executing a `SELECT` query.
It represents a table of data where each row corresponds to a record from the query result.
`ResultSet` allows navigation through rows and retrieval of column values using methods like `getString()`, `getInt()`, etc.

25. How do you navigate through a ResultSet using:

- **next()**

⇒ Moves the cursor to the next row in the `ResultSet`. Returns `true` if there is another row, otherwise `false`.

- **previous()**

⇒ Moves the cursor to the previous row. Works only for scrollable `ResultSet` types.

- **first()**

⇒ Moves the cursor to the first row in the `ResultSet`. Works only for scrollable types.

- **last()**

⇒ Moves the cursor to the last row in the `ResultSet`. Works only for scrollable types.

26. How do you retrieve data using ResultSet?

⇒ Data from the current row of a `ResultSet` is retrieved using getter methods like `getString(columnName)`, `getInt(columnIndex)`, etc.
Column names or indexes can be used to access the values.
Example: `rs.getString("name")` or `rs.getInt(2)` fetches the value from the respective column of the current row.

Lab Questions:

27. Write a Java program to execute a SELECT query and process ResultSet.

⇒

```
import java.sql.*;

public class SelectExample {
    public static void main(String[] args) {
        try {
            // 1. Load Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Establish Connection
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // 3. Create Statement
            Statement stmt = conn.createStatement();

            // 4. Execute SELECT query
            ResultSet rs = stmt.executeQuery("SELECT id, name, age FROM students");

            // 5. Process ResultSet
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                int age = rs.getInt("age");

                System.out.println(id + " | " + name + " | " + age);
            }

            // 6. Close resources
            rs.close();
```

```
            stmt.close();
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

28. Demonstrate ResultSet navigation using next(), previous(), etc.

⇒

```
import java.sql.*;

public class ResultSetNavigation {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // Create scrollable ResultSet
            Statement stmt = conn.createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

            ResultSet rs = stmt.executeQuery("SELECT id, name FROM students");

            // Move to first row
            if (rs.first()) {
                System.out.println("First: " + rs.getInt("id") + " - " + rs.getString("name"));
            }

            // Move to last row
            if (rs.last()) {
                System.out.println("Last: " + rs.getInt("id") + " - " + rs.getString("name"));
            }

            // Move backwards
            if (rs.previous()) {
                System.out.println("Previous: " + rs.getInt("id") + " - " +
```

```
rs.getString("name"));
        }

        // Iterate forward with next()
        rs.beforeFirst();
        while (rs.next()) {
           System.out.println("Next: " + rs.getInt("id") + " - " + rs.getString("name"));
        }

        rs.close();
        stmt.close();
        conn.close();

     } catch (Exception e) {
        e.printStackTrace();
     }
  }
}
```

---

7. Database Metadata

Theory Questions:

29. What is DatabaseMetaData in JDBC?

⇒`DatabaseMetaData` is an interface in JDBC that provides information about the database and its supported features.

It is obtained by calling the `getMetaData()` method of the `Connection` object.

It allows you to query details like database name, version, supported SQL features, tables, and driver information.

30. Why is DatabaseMetaData important?

⇒ `DatabaseMetaData` is important because it helps developers write database-independent programs.

It provides runtime details about the database, enabling the application to adapt to different DBMS without code changes.

It is also useful for dynamically retrieving schema, table structures, and supported SQL features.

31. List and explain some methods of DatabaseMetaData like:

- **getDatabaseProductName**

⇒`getDatabaseProductName()` – Returns the name of the database product, e.g., "MySQL", "Oracle".

- **getTables**

⇒`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)` – Retrieves a description of tables in the database, including table names, types, and schemas.

Lab Questions:

32. Write a Java program to display:

- **Database name**

⇒

```
import java.sql.*;

public class DatabaseNameExample {
    public static void main(String[] args) {
        try {
            // 1. Load Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Establish Connection
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // 3. Get DatabaseMetaData
            DatabaseMetaData metaData = conn.getMetaData();

            // 4. Display Database Name
            System.out.println("Database Name: " + metaData.getDatabaseProductName());

            // 5. Close connection
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
```

```
    }
  }
}
```

- **Version**

⇒

```
import java.sql.*;

public class DatabaseVersionExample {
   public static void main(String[] args) {
      try {
         // 1. Load Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish Connection
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // 3. Get DatabaseMetaData
         DatabaseMetaData metaData = conn.getMetaData();

         // 4. Display Database Version
         System.out.println("Database Version: " +
metaData.getDatabaseProductVersion());

         // 5. Close connection
         conn.close();

      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

- **Tables list**

⇒

```
import java.sql.*;

public class TablesListExample {
   public static void main(String[] args) {
      try {
         // 1. Load Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish Connection
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // 3. Get DatabaseMetaData
         DatabaseMetaData metaData = conn.getMetaData();

         // 4. Get tables list
         ResultSet tables = metaData.getTables(null, null, "%", new String[]{"TABLE"});

         System.out.println("Tables in the database:");
         while (tables.next()) {
            System.out.println(tables.getString("TABLE_NAME"));
         }

         // 5. Close resources
         tables.close();
         conn.close();

      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

- **Supported SQL features using DatabaseMetaData**

⇒

```
import java.sql.*;

public class SupportedSQLFeaturesExample {
```

```
   public static void main(String[] args) {
      try {
         // 1. Load Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish Connection
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // 3. Get DatabaseMetaData
         DatabaseMetaData metaData = conn.getMetaData();

         // 4. Display Supported SQL Features
         System.out.println("Supported SQL Features:");
         System.out.println("Supports Batch Updates: " +
metaData.supportsBatchUpdates());
         System.out.println("Supports Stored Procedures: " +
metaData.supportsStoredProcedures());
         System.out.println("Supports Transactions: " +
metaData.supportsTransactions());
         System.out.println("Supports Full Outer Joins: " +
metaData.supportsFullOuterJoins());
         System.out.println("Supports Group By: " + metaData.supportsGroupBy());

         // 5. Close connection
         conn.close();

      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

---

8. ResultSet Metadata

Theory Questions:

33. What is ResultSetMetaData?

⇒ `ResultSetMetaData` is an interface in JDBC that provides information about the columns in a `ResultSet` object, such as column names, types, and count.
It is obtained by calling the `getMetaData()` method on a `ResultSet`.

34. Why is ResultSetMetaData important?

⇒ `ResultSetMetaData` is important because it allows dynamic handling of query results without hardcoding column details.

It helps when the number, names, or types of columns are unknown at compile time.

35. Explain the use of methods:

- **getColumnCount**

⇒ Returns the total number of columns in the `ResultSet`.

- **getColumnName**

⇒ Returns the name of the specified column by index (1-based).

- **getColumnType**

⇒ Returns the SQL type of the specified column as an integer constant from `java.sql.Types`.

Lab Questions:

36. Write a Java program using ResultSetMetaData to display:

- **Column names**

⇒

```
import java.sql.*;

public class ColumnCountExample {
   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         Statement stmt = conn.createStatement();
```

```
            ResultSet rs = stmt.executeQuery("SELECT * FROM students");

            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();

            System.out.println("Total Columns: " + columnCount);

            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Column types**

⇒

```
import java.sql.*;

public class ColumnNamesExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM students");

            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();

            System.out.println("Column Names:");
            for (int i = 1; i <= columnCount; i++) {
                System.out.println(rsmd.getColumnName(i));
            }

            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
```

```
    }
  }
}
```

- **Column count**

⇒

```
import java.sql.*;

public class ColumnTypesExample {
   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT * FROM students");

         ResultSetMetaData rsmd = rs.getMetaData();
         int columnCount = rsmd.getColumnCount();

         System.out.println("Column Types:");
         for (int i = 1; i <= columnCount; i++) {
            System.out.println(rsmd.getColumnTypeName(i));
         }

         conn.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

9. Practical SQL Query Examples

Lab Questions:

37. Write SQL queries to:

- **Insert a record into a table**

⇒

```
INSERT INTO students (id, name, age) VALUES (1, 'Ayush', 21);
```

- **Update specific fields**

⇒

```
UPDATE students SET age = 22 WHERE id = 1;
```

- **Select records based on condition**

⇒

```
SELECT * FROM students WHERE age > 20;
```

- **Delete specific records**

⇒

```
DELETE FROM students WHERE id = 1;
```

38. Implement the above queries in a Java JDBC program.

```
import java.sql.*;

public class JDBC_CRUD_Example {
   public static void main(String[] args) {
      try {
         // 1. Load JDBC Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish Connection
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // ---------------- INSERT ----------------
         String insertSQL = "INSERT INTO students (id, name, age) VALUES (?, ?, ?)";
         PreparedStatement insertStmt = conn.prepareStatement(insertSQL);
         insertStmt.setInt(1, 1);
         insertStmt.setString(2, "Ayush");
         insertStmt.setInt(3, 21);
         insertStmt.executeUpdate();
         System.out.println("Record Inserted.");

         // ---------------- UPDATE ----------------
         String updateSQL = "UPDATE students SET age = ? WHERE id = ?";
         PreparedStatement updateStmt = conn.prepareStatement(updateSQL);
         updateStmt.setInt(1, 22);
         updateStmt.setInt(2, 1);
         updateStmt.executeUpdate();
         System.out.println("Record Updated.");

         // ---------------- SELECT ----------------
         String selectSQL = "SELECT * FROM students WHERE age > ?";
         PreparedStatement selectStmt = conn.prepareStatement(selectSQL);
         selectStmt.setInt(1, 20);
         ResultSet rs = selectStmt.executeQuery();
         System.out.println("Selected Records:");
         while (rs.next()) {
            System.out.println(rs.getInt("id") + " | " +
                               rs.getString("name") + " | " +
                               rs.getInt("age"));
         }

         // ---------------- DELETE ----------------
         String deleteSQL = "DELETE FROM students WHERE id = ?";
         PreparedStatement deleteStmt = conn.prepareStatement(deleteSQL);
```

```
            deleteStmt.setInt(1, 1);
            deleteStmt.executeUpdate();
            System.out.println("Record Deleted.");

            // Close resources
            rs.close();
            insertStmt.close();
            updateStmt.close();
            selectStmt.close();
            deleteStmt.close();
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

10. Practical Example 1: Swing GUI for CRUD Operations

Theory Questions:

39. What is Java Swing?

⇒ Java Swing is a GUI toolkit in Java used to create desktop applications.

It provides components like buttons, text fields, labels, tables, and frames for building interactive UIs.

40. How can we integrate Swing with JDBC for database operations?

⇒ Swing can be integrated with JDBC by attaching event listeners to UI components (e.g., buttons) and performing database operations inside those event handlers.

For example, clicking an "Insert" button triggers JDBC code to insert data into the database.

Lab Questions:

41. Create a Swing GUI with input fields for id, fname, lname, and email.

⇒

```
import javax.swing.*;
import java.awt.*;

public class SwingFormExample extends JFrame {
   JTextField tfId, tfFname, tfLname, tfEmail;

   public SwingFormExample() {
      setTitle("User Form");
      setSize(300, 200);
      setLayout(new GridLayout(4, 2, 5, 5));

      // Labels and Text Fields
      add(new JLabel("ID:"));
      tfId = new JTextField();
      add(tfId);

      add(new JLabel("First Name:"));
      tfFname = new JTextField();
      add(tfFname);

      add(new JLabel("Last Name:"));
      tfLname = new JTextField();
      add(tfLname);

      add(new JLabel("Email:"));
      tfEmail = new JTextField();
      add(tfEmail);

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
   }

   public static void main(String[] args) {
      new SwingFormExample();
   }
}
```

42. Implement Insert, Update, Select, Delete operations on button clicks using JDBC.

⇒

```
import javax.swing.*;
import java.awt.*;
```

```
import java.sql.*;

public class SwingCRUD extends JFrame {
   JTextField tfId, tfFname, tfLname, tfEmail;
   JButton btnInsert, btnUpdate, btnSelect, btnDelete;
   Connection conn;

   public SwingCRUD() {
      setTitle("Swing JDBC CRUD Operations");
      setSize(400, 250);
      setLayout(new GridLayout(6, 2, 5, 5));

      // Input fields
      add(new JLabel("ID:")); tfId = new JTextField(); add(tfId);
      add(new JLabel("First Name:")); tfFname = new JTextField(); add(tfFname);
      add(new JLabel("Last Name:")); tfLname = new JTextField(); add(tfLname);
      add(new JLabel("Email:")); tfEmail = new JTextField(); add(tfEmail);

      // Buttons
      btnInsert = new JButton("Insert");
      btnUpdate = new JButton("Update");
      btnSelect = new JButton("Select");
      btnDelete = new JButton("Delete");

      add(btnInsert); add(btnUpdate);
      add(btnSelect); add(btnDelete);

      // Database connection
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");
      } catch (Exception e) {
         JOptionPane.showMessageDialog(this, "DB Connection Failed: " +
e.getMessage());
      }

      // Action listeners
      btnInsert.addActionListener(e -> insertRecord());
      btnUpdate.addActionListener(e -> updateRecord());
      btnSelect.addActionListener(e -> selectRecord());
      btnDelete.addActionListener(e -> deleteRecord());

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
   }
```

```
void insertRecord() {
   try {
      String sql = "INSERT INTO users (id, fname, lname, email) VALUES (?, ?, ?, ?)";
      PreparedStatement pst = conn.prepareStatement(sql);
      pst.setInt(1, Integer.parseInt(tfId.getText()));
      pst.setString(2, tfFname.getText());
      pst.setString(3, tfLname.getText());
      pst.setString(4, tfEmail.getText());
      pst.executeUpdate();
      JOptionPane.showMessageDialog(this, "Record Inserted");
   } catch (Exception ex) {
      JOptionPane.showMessageDialog(this, ex.getMessage());
   }
}

void updateRecord() {
   try {
      String sql = "UPDATE users SET fname=?, lname=?, email=? WHERE id=?";
      PreparedStatement pst = conn.prepareStatement(sql);
      pst.setString(1, tfFname.getText());
      pst.setString(2, tfLname.getText());
      pst.setString(3, tfEmail.getText());
      pst.setInt(4, Integer.parseInt(tfId.getText()));
      pst.executeUpdate();
      JOptionPane.showMessageDialog(this, "Record Updated");
   } catch (Exception ex) {
      JOptionPane.showMessageDialog(this, ex.getMessage());
   }
}

void selectRecord() {
   try {
      String sql = "SELECT * FROM users WHERE id=?";
      PreparedStatement pst = conn.prepareStatement(sql);
      pst.setInt(1, Integer.parseInt(tfId.getText()));
      ResultSet rs = pst.executeQuery();
      if (rs.next()) {
         tfFname.setText(rs.getString("fname"));
         tfLname.setText(rs.getString("lname"));
         tfEmail.setText(rs.getString("email"));
      } else {
         JOptionPane.showMessageDialog(this, "Record Not Found");
      }
   } catch (Exception ex) {
      JOptionPane.showMessageDialog(this, ex.getMessage());
```

```
    }
}

void deleteRecord() {
    try {
        String sql = "DELETE FROM users WHERE id=?";
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setInt(1, Integer.parseInt(tfId.getText()));
        pst.executeUpdate();
        JOptionPane.sh
```

---

11. Practical Example 2: CallableStatement with IN and OUT Parameters

Theory Questions:

43. What is CallableStatement in JDBC?

⇒ `CallableStatement` is an interface in JDBC used to execute stored procedures in a database.
It is obtained using the `prepareCall()` method of the `Connection` object.

44. How to use CallableStatement to call stored procedures?

⇒ To use `CallableStatement`, first create a stored procedure in the database, then use `prepareCall()` with the procedure's SQL syntax, set the IN/OUT parameters, execute it, and retrieve results if needed.

45. What are IN and OUT parameters in stored procedures?

⇒ IN parameter – Accepts a value from the calling program to the stored procedure.

⇒ OUT parameter – Returns a value from the stored procedure back to the calling program.

Lab Questions:

46. Create a stored procedure in MySQL with IN and OUT parameters.

⇒

```
DELIMITER //

CREATE PROCEDURE getStudentNameById(
   IN student_id INT,
   OUT student_name VARCHAR(50)
)
BEGIN
   SELECT fname INTO student_name
   FROM users
   WHERE id = student_id;
END //

DELIMITER ;
```

47. Write a Java program using CallableStatement to call the stored procedure.

⇒

```
import java.sql.*;

public class CallableStatementExample {
   public static void main(String[] args) {
      try {
         // 1. Load JDBC Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish Connection
         Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // 3. Prepare CallableStatement for stored procedure
         CallableStatement cstmt = conn.prepareCall("{CALL getStudentNameById(?,
?)}");

         // 4. Set IN parameter
         cstmt.setInt(1, 1); // ID = 1

         // 5. Register OUT parameter
         cstmt.registerOutParameter(2, Types.VARCHAR);

         // 6. Execute stored procedure
```

```
            cstmt.execute();

            // 7. Retrieve OUT parameter value
            String studentName = cstmt.getString(2);
            System.out.println("Student Name: " + studentName);

            // 8. Close resources
            cstmt.close();
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

48. Demonstrate passing IN and retrieving OUT parameters.

⇒

```
import java.sql.*;

public class INOUT_Demo {
    public static void main(String[] args) {
        try {
            // Load JDBC Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Connect to Database
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // Prepare CallableStatement
            CallableStatement cstmt = conn.prepareCall("{CALL getStudentNameById(?,
?)}");

            // Pass IN parameter (Student ID)
            cstmt.setInt(1, 1);

            // Register OUT parameter (Student Name)
            cstmt.registerOutParameter(2, Types.VARCHAR);
```

```
            // Execute Stored Procedure
            cstmt.execute();

            // Retrieve OUT parameter value
            String name = cstmt.getString(2);
            System.out.println("Student Name: " + name);

            // Close resources
            cstmt.close();
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```