Module 7 – RDBMS & Database Programming with JDBC

1. Introduction to JDBC

Que: 1

What is JDBC (Java Database Connectivity)?

Ans:

❖ **Meaning:**

JDBC is a Java API used to connect Java applications with databases and execute SQL statements.

➢ **Why JDBC is used:**

JDBC is used to:

- Connect Java application with a database (MySQL, Oracle, PostgreSQL, etc.)
- Perform CRUD operations
- (Create, Read, Update, Delete)
- Execute SQL queries from Java
- Retrieve and process data from the database

➢ **JDBC Architecture (How it works)**

- Java Application
- JDBC API
- JDBC Driver
- Database
  - **Java app → JDBC API → JDBC Driver → Database**

➢ **Main Components of JDBC**

1) **DriverManager**

   Manages database drivers

   Establishes database connection

2) **Connection**

   Represents a connection to the database

3) **Statement / PreparedStatement / CallableStatement**

   Used to execute SQL queries

4) **ResultSet**

   Stores the result of SELECT queries

➢ **Steps to Use JDBC**

- Load JDBC Driver
- Create Connection
- Create Statement
- Execute SQL Query
- Process ResultSet
- Close Connection

➢ **Types of JDBC Drivers**

- Type 1 – JDBC-ODBC Bridge (Deprecated)
- Type 2 – Native API Driver
- Type 3 – Network Protocol Driver
- Type 4 – Thin Driver (Most commonly used)

➢ **Advantages of JDBC**

- Platform independent
- Supports multiple databases
- Secure and reliable
- Easy integration with Java applications

Que: 2

Importance of JDBC in Java Programming

Ans:

JDBC is very important in Java because it acts as a bridge between Java applications and databases. Almost all real-world Java applications (banking, e-commerce, hospital, ERP systems) use JDBC.

❖ **Key Importance of JDBC**

1. Database Connectivity

- JDBC allows Java applications to connect with databases like MySQL, Oracle, PostgreSQL
- Without JDBC, Java cannot communicate with databases

2. Perform CRUD Operations

- **JDBC enables:**

  Create → Insert data

  Read → Fetch data

  Update → Modify data

  Delete → Remove data

- These operations are essential for any application.

3. Platform Independent

- Java + JDBC works on any OS
- Same Java code can connect to different databases by changing the driver

4. Supports Multiple Databases

- One Java program can work with different databases
- Only JDBC driver changes, not the Java code

5. Secure Database Access

- Supports PreparedStatement to prevent SQL Injection

- Helps in writing secure applications

6. Efficient Performance

- PreparedStatement improves performance
- Reduces query compilation overhead

7. Widely Used in Enterprise Applications

- **Used in:**

  Banking systems

  Web applications

  Android apps

  Spring / Hibernate based projects

8. Transaction Management

- Supports commit, rollback
- Ensures data consistency (important in money transfer systems)

9. Foundation for Advanced Technologies

- Frameworks like Hibernate, JPA, Spring JDBC internally use JDBC
- Learning JDBC is mandatory before learning these frameworks

Que: 3

JDBC Architecture: Driver Manager, Driver, Connection, Statement, and ResultSet

Ans:

- **JDBC Architecture**

  JDBC architecture defines how a Java application communicates with a database using JDBC components.

- **Flow:**

  Java Application → JDBC API → JDBC Driver → Database

1. **DriverManager**
   - **Role:**
     - Manages JDBC drivers
     - Establishes connection between Java application and database
   - **Key Points:**
     - Loads database driver
     - Provides getConnection() method
     - Acts as a factory for Connection
   - **Example:**

     ```
     Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/testdb", "root", "password");
     ```
   - **Interview Line:**

     DriverManager is a class that manages JDBC drivers and creates database connections.
2. **Driver**
   - **Role:**
     - Database-specific implementation
     - Converts JDBC calls into database-specific protocol
   - **Key Points:**
     - Provided by database vendors
     - MySQL → **com.mysql.jdbc.Driver**

    - Oracle → oracle.jdbc.driver.OracleDriver

- **Interview Line:**

  Driver is a database-specific class that handles communication between Java and the database.

3. **Connection**
   - **Role:**
     - Represents an active connection to the database
   - **Key Points:**
     - Used to create Statement objects
     - Manages transactions
     - Must be closed after use
   - **Example:**

     ```
     Connection con = DriverManager.getConnection(url, user, pass);
     ```

   - **Interview Line:**

     Connection is an interface that represents a session between Java application and database.

4. **Statement**
   - **Role:**
     - Executes SQL queries
   - **Types of Statement:**

     1. Statement – Simple SQL

     2. PreparedStatement – Precompiled, secure

     3. CallableStatement – Stored procedures

   - **Example:**

     ```
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM student");
     ```

   - **Interview Line:**

     Statement is used to send SQL commands to the database.

5. **ResultSet**

- **Role:**
    - Stores data returned from SELECT query
- **Key Points:**
    - Works like a cursor
    - Provides methods like next(), getInt(), getString()
- **Example:**

```
while(rs.next()) {
   System.out.println(rs.getInt(1) + " " + rs.getString(2));
}
```

- **Interview Line:**

  ResultSet is an interface that holds the result of a database query.

2. JDBC Driver Types

Que: 1

o Overview of JDBC Driver Types:

Type 1: JDBC-ODBC Bridge Driver

Type 2: Native-API Driver

Type 3: Network Protocol Driver

Type 4: Thin Driver

Ans:

❖ **Overview of JDBC Driver Types:**

Type 1 – JDBC-ODBC Bridge Driver: Uses ODBC driver to connect Java with database; not fully Java, slower, mostly obsolete.

Type 2 – Native-API Driver: Uses database's native API; faster than Type 1 but platform-dependent.

Type 3 – Network Protocol Driver: Uses a middleware server to translate calls to database; platform-independent.

Type 4 – Thin Driver: Pure Java driver that directly communicates with the database; fast and portable.

❖ **Advantages and Disadvantages for the 4 JDBC driver types:**

Driver Type	**Advantages**	**Disadvantages**
Type 1 – JDBC-ODBC Bridge	Easy to use; works with any ODBC data source	Slow; requires ODBC setup; platform-dependent; obsolete
Type 2 – Native-API Driver	Faster than Type 1; uses database features efficiently	Platform-dependent; needs native library on client machine
Type 3 – Network Protocol Driver	Platform-independent; good for internet applications	Requires middleware server; extra network layer may slow performance
Type 4 – Thin Driver	Pure Java; fast; platform-independent; no extra software needed	Database-specific; minor compatibility issues may occur

Que: 2

Comparison and Usage of Each Driver Type

Ans:

❖ **Comparison of JDBC Driver Types:**

Feature	**Type 1 JDBC-ODBC Bridge**	**Type 2 Native-API**	**Type 3 Network Protocol**	**Type 4 Thin Driver**
Language	Java + ODBC	Java + Native code	Pure java	Pure java
Platform Dependent	Yes	Yes	No	No
Middleware	No	No	Yes	No
Native Libraries Required	Yes (ODBC)	Yes	No	No
Performance	Low	Medium	Medium	High
DB Independence	Yes	No	Yes	No
Web Application Support	No	No	Yes	Yes
Security	Low	Medium	High	High
Current Usage	Deprecated	Rare	Limited	Most popular

- **Usage of Each JDBC Driver Type**

  Type 1: JDBC-ODBC Bridge Driver

  - **Used When:**
    - Small desktop applications
    - Learning or testing purpose
    - Database only provides ODBC driver
  - **Not Used Because:**
    - Slow performance
    - Requires ODBC installation
    - Deprecated after Java 8

  Type 2: Native-API Driver

  - **Used When:**

    - Legacy systems
    - Performance is important
    - Native DB libraries are already installed
- **Limitations:**
    - Platform dependent
    - Difficult deployment

Type 3: Network Protocol Driver

- **Used When:**
    - Large enterprise applications
    - Need to connect multiple databases
    - Centralized middleware management required
- **Limitations:**
    - Extra middleware setup
    - Performance depends on network

Type 4: Thin Driver

- **Used When:**
    - Web applications
    - Enterprise applications
    - Modern Java projects
- **Why Most Popular:**
    - Best performance
    - No native code
    - Easy deployment
    - Platform independent

- **Examples:**

MySQL → com.mysql.cj.jdbc.Driver

Oracle → oracle.jdbc.driver.OracleDriver

3. Steps for Creating JDBC Connections

Que: 1

Step-by-Step Process to Establish a JDBC Connection:

1. Import the JDBC packages

2. Register the JDBC driver

3. Open a connection to the database

4. Create a statement

5. Execute SQL queries

6. Process the resultset

7. Close the connection

Ans:

❖ **Step-by-Step Process to Establish a JDBC Connection**

1. **Import the JDBC Packages**
    - **Purpose:**

      To use JDBC classes and interfaces such as Connection, Statement, ResultSet.
    - **Code:**

      ```
      import java.sql.*;
      ```
2. **Register the JDBC Driver**
    - **Purpose:**

      To load the database driver so Java can communicate with the database.
    - **Ways to Register Driver:**

      (a) Using Class.forName() (Old way):

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

      (b) Automatic Driver Loading (Recommended):

      From JDBC 4.0 onward, drivers are auto-loaded when JAR is added to classpath.

3. **Open a Connection to the Database**

- **Purpose:**

  Establishes a connection between Java application and database.

- **Code:**

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mydb",
    "username",
    "password"
);
```

4. **Create a Statement**

- **Purpose:**

  Used to send SQL queries to the database.

- **Types of Statements:**

  Statement

  PreparedStatement

  CallableStatement

- **Code:**

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

  (Preferred – PreparedStatement)

```
PreparedStatement ps =
    con.prepareStatement("SELECT * FROM student WHERE id=?");
```

5. **Execute SQL Queries**

- **Purpose:**

  Runs SQL commands (SELECT, INSERT, UPDATE, DELETE).

- **Methods Used:**

executeQuery() → SELECT

executeUpdate() → INSERT, UPDATE, DELETE

execute() → Any SQL

- **Code:**

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

6. **Process the ResultSet**

- **Purpose:**

  Reads and processes the data returned from the database.

- **Code:**

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

7. **Close the Connection**

- **Purpose:**

  Releases database resources and avoids memory leaks.

- **Order of Closing:**

  ResultSet

  Statement

  Connection

- **Code:**

```
rs.close();

stmt.close();
```

```
     con.close();
```

- **Complete JDBC Example**

```
import java.sql.*;


public class JDBCExample {
    public static void main(String[] args) {


        try {
            // Register Driver (optional in new versions)
            Class.forName("com.mysql.cj.jdbc.Driver");


            // Create Connection
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/mydb",
                "root",
                "password"
            );


            // Create Statement
            Statement stmt = con.createStatement();


            // Execute Query
            ResultSet rs = stmt.executeQuery("SELECT * FROM student");


            // Process ResultSet
            while (rs.next()) {
                System.out.println(
                    rs.getInt(1) + " " + rs.getString(2)
                );
            }

            // Close Connection
            con.close();

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

4. Types of JDBC Statements

Que: 1

Overview of JDBC Statements:

Statement: Executes simple SQL queries without parameters.

PreparedStatement: Precompiled SQL statements for queries with parameters.

CallableStatement: Used to call stored procedures.

Ans:

❖ **Overview of JDBC Statements**

In JDBC, Statements are used to send SQL commands from a Java application to the database.

- **JDBC provides three types of Statement interfaces:**

  Statement

  PreparedStatement

  CallableStatement

1. **Statement**
   - **Description:**
     - Used to execute simple SQL queries without parameters.
     - SQL query is compiled every time it is executed.
   - **Key Features:**
     - No parameters supported
     - Slower than PreparedStatement
     - More prone to SQL Injection
   - **Example:**

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

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

- **When to Use:**
  - Static queries

  - Simple SELECT operations
  - Testing or learning purpose

2. **PreparedStatement**
   - **Description:**
     - Used for SQL queries with parameters (?).
     - SQL query is precompiled and stored in the database.
   - **Key Features:**
     - Faster performance
     - Prevents SQL Injection
     - Supports dynamic values
   - **Example:**

```
PreparedStatement ps =
   con.prepareStatement("SELECT * FROM student WHERE id = ?");
ps.setInt(1, 101);
ResultSet rs = ps.executeQuery();
```

   - **When to Use:**
     - Repeated queries
     - Queries with user input
     - Production & enterprise applications

3. **CallableStatement**
   - **Description:**
     - Used to call stored procedures from the database.
     - Can accept IN, OUT, and INOUT parameters.
   - **Key Features:**
     - Executes stored procedures
     - Improves performance and security
     - Business logic handled in database
   - **Example:**

```
CallableStatement cs =
    con.prepareCall("{call getStudent(?)}");

cs.setInt(1, 101);

ResultSet rs = cs.executeQuery();
```

- **When to Use:**
  - Complex database logic
  - Stored procedures
  - Enterprise applications
  - Comparison of JDBC Statements

- **Interview Tips**
  - **Most recommended:** PreparedStatement
  - **Fastest & secure:** PreparedStatement
  - **Stored procedure support:** CallableStatement
  - **Least used:** Statement

Que: 2

Differences between Statement, PreparedStatement, and CallableStatement.

Ans:

❖ **Differences between Statement, PreparedStatement, and CallableStatement**

Feature	**Statement**	**PreparedStatement**	**CallableStatement**
Package	Java.sql	Java.sql	Java.sql
Purpose	Execute simple SQL queries	Execute parameterized SQL queries	Call stored procedures
Parameters Support	No	Yes (?)	Yes (IN, OUT, INOUT)
Precompiled	No	Yes	Yes
Performance	Low	High	Very high
SQL Injection	Vulnerable	Prevented	Prevented
Reusability	No	Yes	Yes
Stored Procedure	No	No	Yes
Complexity	Simple	Medium	Advanced
Most Used In	Learning/Testing	Real – time applications	Enterprise applications

➢ **Code Examples**

- **Statement**

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

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

- **PreparedStatement**

```
PreparedStatement ps =

    con.prepareStatement("SELECT * FROM student WHERE id=?");

ps.setInt(1, 101);

ResultSet rs = ps.executeQuery();
```

- **CallableStatement**

```
CallableStatement cs =
    con.prepareCall("{call getStudent(?)}");
cs.setInt(1, 101);
ResultSet rs = cs.executeQuery();
```

- **When to Use Which?**

- **Statement**

  → Simple, static queries (rarely used)

- **PreparedStatement**

  → Best choice for most applications

  → Secure, fast, and reusable

- **CallableStatement**

  → When working with stored procedures

- **Interview One-Line Answers**
  - **Statement:** Executes simple SQL without parameters.
  - **PreparedStatement:** Executes precompiled SQL with parameters.
  - **CallableStatement:** Used to call stored procedures.

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

Que: 1

o Insert: Adding a new record to the database.

o Update: Modifying existing records.

o Select: Retrieving records from the database.

o Delete: Removing records from the database.

Ans:

1. **Insert: Adding a new record to the database.**
   - **SQL Syntax**

     ```
     INSERT INTO student (id, name, age) VALUES (101, 'Rahul', 22);
     ```

   - **JDBC Example**

     ```
     PreparedStatement ps =
         con.prepareStatement("INSERT INTO student VALUES (?, ?, ?)");
     ps.setInt(1, 101);
     ps.setString(2, "Rahul");
     ps.setInt(3, 22);
     int result = ps.executeUpdate();
     ```

2. **Update: Modifying existing records.**
   - **SQL Syntax**

     ```
     UPDATE student SET age = 23 WHERE id = 101;
     ```

   - **JDBC Example**

     ```
     PreparedStatement ps =
         con.prepareStatement("UPDATE student SET age=? WHERE id=?");
     ps.setInt(1, 23);
     ps.setInt(2, 101);
     int result = ps.executeUpdate();
     ```

3. **Select: Retrieving records from the database.**
   - **SQL Syntax**

     ```
     SELECT * FROM student;
     ```
   - **JDBC Example**

     ```
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM student");

     while (rs.next()) {
         System.out.println(
             rs.getInt("id") + " " +
             rs.getString("name") + " " +
             rs.getInt("age")
         );
     }
     ```
4. **Delete: Removing records from the database.**
   - **SQL Syntax**

     ```
     DELETE FROM student WHERE id = 101;
     ```
   - **JDBC Example**

     ```
     PreparedStatement ps =
         con.prepareStatement("DELETE FROM student WHERE id=?");
     ps.setInt(1, 101);

     int result = ps.executeUpdate();
     ```

❖ **Important JDBC Methods Used**

Method	**Used For**
executeUpdate()	INSERT, UPDATE, DELETE
executeQuery()	SELECT
execute()	Any SQL

6. ResultSet Interface

Que: 1

What is ResultSet in JDBC?

Ans:

❖ **Meaning:**

ResultSet is an interface in the java.sql package that represents the data returned by a SELECT SQL query in JDBC.

- It stores database records in row and column (tabular) format.
- It acts like a cursor that points to one row at a time.
- It allows Java programs to read and process data from the database.

➢ **How ResultSet is Obtained**

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

➢ **Key Points**

- Initially, the cursor is positioned before the first row.
- rs.next() moves the cursor to the next row.
- Data can be retrieved using column index or column name.

➢ **Example**

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

One-Line Interview Answer

ResultSet is an interface in JDBC used to store and retrieve data returned by a SELECT query in tabular form.

Que: 2

Navigating through ResultSet (first, last, next, previous)

Ans:

❖ **Navigating through ResultSet in JDBC**

- In JDBC, a ResultSet uses a cursor to move through rows returned by a SELECT query.
- By default, the cursor is positioned before the first row.

➢ **Important ResultSet Navigation Methods**

1. **next()**
   - Moves the cursor to the next row
   - Most commonly used method
   - Example:

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

2. **first()**
   - Moves the cursor to the first row
   - Works only with scrollable ResultSet
   - Example:

   ```
   rs.first();
   System.out.println(rs.getString("name"));
   ```

3. **last()**
   - Moves the cursor to the last row
   - Works only with scrollable ResultSet
   - Example:

   ```
   rs.last();
   System.out.println(rs.getInt("id"));
   ```

4. **previous()**
   - Moves the cursor to the previous row

- Works only with scrollable ResultSet
- Example;

```
rs.previous();

System.out.println(rs.getString("name"));
```

- **Creating a Scrollable ResultSet (Required for first, last, previous)**

```
Statement stmt = con.createStatement(

   ResultSet.TYPE_SCROLL_INSENSITIVE,

   ResultSet.CONCUR_READ_ONLY

);


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

- **Interview Tip**
  - **next()** works with all ResultSet types
  - **first(), last(), previous()** need TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE
  - **ResultSet** navigation methods like next, first, last, and previous are used to move the cursor through rows returned by a SQL query.

Que: 3

Working with ResultSet to retrieve data from SQL queries

Ans:

- **Working with ResultSet to Retrieve Data from SQL Queries**
  - In JDBC, ResultSet is used to read and process data returned by a SELECT query.
  - It allows access to each row and column using getter methods.
- **Step-by-Step Process**

1. **Execute SELECT Query**
   - A ResultSet is obtained using executeQuery().

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

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

2. **Move the Cursor**
   - The cursor starts before the first row
   - next() moves it to the next row

   ```
   while (rs.next()) {

      // process each row

   }
   ```

3. **Retrieve Data from Columns**
   - **Data can be retrieved using:**
     - Column index (starts from 1)
     - Column name
   - **Using Column Index**

   ```
   int id = rs.getInt(1);

   String name = rs.getString(2);
   ```

   - **Using Column Name**

   ```
   int id = rs.getInt("id");

   String name = rs.getString("name");
   ```

- **Common ResultSet Getter Methods**

Method	**Data Type**
getInt()	Integer
getString()	String
getDouble()	Double
getBoolean()	Boolean
getDate()	Date

- **Complete Example**

```
while (rs.next()) {

   int id = rs.getInt("id");

   String name = rs.getString("name");

   int age = rs.getInt("age");



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

}
```

- **Using PreparedStatement with ResultSet**

```
PreparedStatement ps =

   con.prepareStatement("SELECT * FROM student WHERE id=?");

ps.setInt(1, 101);



ResultSet rs = ps.executeQuery();



if (rs.next()) {

   System.out.println(rs.getString("name"));

}
```

7. Database Metadata

Que: 1

What is DatabaseMetaData?

Ans:

- **Meaning:**

  DatabaseMetaData is an interface in the java.sql package that provides information about the database itself, not the actual data stored in tables.

  It gives details such as:

  - Database name and version
  - Supported SQL features
  - Tables, views, schemas
  - Drivers and connection capabilities

- **How to Get DatabaseMetaData**

  ```
  Connection con = DriverManager.getConnection(url, user, pass);

  DatabaseMetaData dbmd = con.getMetaData();
  ```

Que: 2.

Importance of Database Metadata in JDBC

Ans:

Database metadata is important because it helps developers:

- Write database-independent applications
- Check database capabilities at runtime
- Retrieve table, column, and schema information dynamically
- Improve portability and flexibility of applications
- Perform DB analysis and reporting
- Useful for tools like ORM frameworks and admin panels

Que: 3

Methods Provided by DatabaseMetaData (getDatabaseProductName, getTables, etc.)

Ans:

❖ **Commonly Used Methods**

1. **getDatabaseProductName()**
   - Returns the name of the database.

```
System.out.println(dbmd.getDatabaseProductName());
```

2. **getDatabaseProductVersion()**
   - Returns database version.

```
System.out.println(dbmd.getDatabaseProductVersion());
```

3. **getDriverName()**
   - Returns JDBC driver name.

```
System.out.println(dbmd.getDriverName());
```

4. **getDriverVersion()**
   - Returns driver version.

```
System.out.println(dbmd.getDriverVersion());
```

5. **getURL()**
   - Returns database URL.

```
System.out.println(dbmd.getURL());
```

6. **getUserName()**
   - Returns database user name.

```
System.out.println(dbmd.getUserName());
```

7. **getTables()**
   - Returns information about tables in the database.

```
ResultSet rs = dbmd.getTables(null, null, "%", new String[]{"TABLE"});

while (rs.next()) {

   System.out.println(rs.getString("TABLE_NAME"));
}
```

8. **getColumns()**

   - Returns column details of a table.

```
ResultSet rs = dbmd.getColumns(null, null, "student", "%");
while (rs.next()) {
   System.out.println(
      rs.getString("COLUMN_NAME") + " " +
      rs.getString("TYPE_NAME")
   );
}
```

9. **supportsTransactions()**

   - Checks whether DB supports transactions.

```
System.out.println(dbmd.supportsTransactions());
```

8. ResultSet Metadata

Que: 1

What is ResultSetMetaData?

Ans:

- **Meaning:**

  ResultSetMetaData is an interface in the java.sql package that provides information about the structure of the data returned by a SELECT query.

  - It describes the columns of a ResultSet
  - It does not contain actual row data
  - It is useful when column details are not known in advance

- **How to Get ResultSetMetaData**

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

ResultSetMetaData rsmd = rs.getMetaData();
```

Que: 2

Importance of ResultSetMetaData in Analyzing Query Results

Ans:

- **ResultSetMetaData is important because it helps to:**
  - Find the number of columns in a query result
  - Get column names and data types dynamically
  - Build generic and dynamic applications
  - Create reports, table viewers, admin panels
  - Avoid hardcoding column information
  - Useful in frameworks and tools (ORMs, DB utilities)
- **Especially helpful when:**
  - SQL query changes dynamically
  - Tables are not known at compile time

Que: 3

Methods in ResultSetMetaData (getColumnCount, getColumnName, getColumnType)

Ans:

1. **getColumnCount()**
   - Returns the total number of columns in the ResultSet.

   ```
   int count = rsmd.getColumnCount();

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

2. **getColumnName(int column)**
   - Returns the name of the specified column.

   ```
   String colName = rsmd.getColumnName(1);

   System.out.println(colName);
   ```

3. **getColumnType(int column)**
   - Returns the SQL data type of the column (from java.sql.Types).

   ```
   int colType = rsmd.getColumnType(1);

   System.out.println(colType);
   ```

- **Complete Example**

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

ResultSetMetaData rsmd = rs.getMetaData();

int columnCount = rsmd.getColumnCount();


for (int i = 1; i <= columnCount; i++) {

   System.out.println(

      "Column Name: " + rsmd.getColumnName(i) +

      ", Type: " + rsmd.getColumnTypeName(i)

   );

}
```

- **Interview One-Line Answers**
  - **ResultSetMetaData:** Provides column information of a ResultSet
  - **Importance:** Helps analyze query result structure dynamically
  - **Usage:** Used to get column count, name, and data type

10. Swing GUI for CRUD Operations

Que: 1

Introduction to Java Swing for GUI Development

Ans:

❖ **Meaning:**

Java Swing is a GUI (Graphical User Interface) toolkit in Java used to create desktop-based applications.

➢ **Key Points**

- Part of Java Foundation Classes (JFC)
- Written completely in Java (platform independent)
- Provides rich set of GUI components
- Supports event-driven programming
- More powerful and flexible than AWT

➢ **Common Swing Components**

Component	**Purpose**
JFrame	Main window
JPanel	Container
JLabel	Display text
JTextField	Input field
JButton	Click button
JTable	Display tabular data
JOptionPane	Dialog boxes

➢ **Simple Swing Example**

```
import javax.swing.*;

public class SwingDemo {

    public static void main(String[] args) {
        JFrame f = new JFrame("My Swing App");

        JButton b = new JButton("Click");


        f.add(b);

        f.setSize(300, 200);

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        f.setVisible(true);

    }

}
```

Que: 2

How to Integrate Swing Components with JDBC for CRUD Operations

Ans:

- **Swing + JDBC is commonly used to build desktop database applications like:**
  - Student Management System
  - Billing System
  - Inventory System
- **Integration Flow (Swing + JDBC)**

  User Action (Button Click)

  ↓

  Swing Event (ActionListener)

  ↓

  JDBC Code (CRUD Operation)

  ↓

  Database

  ↓

  Result shown in Swing UI

- **CRUD Integration Example**

  1. **INSERT (Create)**

```
btnAdd.addActionListener(e -> {

  try {

      Connection con = DriverManager.getConnection(url, user, pass);

      PreparedStatement ps =

          con.prepareStatement("INSERT INTO student VALUES (?, ?)");


      ps.setInt(1, Integer.parseInt(tfId.getText()));
```

```
        ps.setString(2, tfName.getText());

        ps.executeUpdate();
        JOptionPane.showMessageDialog(null, "Record Inserted");
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
});
```

2. **SELECT (Read)**

```
PreparedStatement ps =
    con.prepareStatement("SELECT * FROM student");
ResultSet rs = ps.executeQuery();

while (rs.next()) {
    System.out.println(rs.getInt(1) + " " + rs.getString(2));
}
```

(Usually displayed in JTable)

3. **UPDATE**

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

ps.setString(1, tfName.getText());
ps.setInt(2, Integer.parseInt(tfId.getText()));
ps.executeUpdate();
```

4. **DELETE**

```
PreparedStatement ps =
   con.prepareStatement("DELETE FROM student WHERE id=?");


ps.setInt(1, Integer.parseInt(tfId.getText()));
ps.executeUpdate();
```

- **Interview One-Line Answers**
  - **Java Swing:** A GUI toolkit used to create desktop-based Java applications
  - **Swing + JDBC:** Swing handles UI, JDBC handles database operations through event handling

11. Callable Statement with IN and OUT Parameters

Que: 1

What is a CallableStatement?

Ans:

- **Meaning:**

  CallableStatement is an interface in the java.sql package used to call stored procedures present in the database from a Java application.

- **Key Points**
  - It is a sub-interface of PreparedStatement
  - Used to execute precompiled stored procedures
  - Supports IN, OUT, and INOUT parameters
  - Improves performance, security, and reusability

- **Syntax**

  ```
  CallableStatement cs = con.prepareCall("{call procedureName(?, ?)}");
  ```

Que: 2

How to Call Stored Procedures Using CallableStatement in JDBC.

Ans:

- **Steps:**
  1. Create database connection
  2. Use prepareCall()
  3. Set IN parameters
  4. Register OUT parameters
  5. Execute stored procedure
  6. Retrieve OUT parameters

- **Example Stored Procedure (MySQL)**

```
CREATE PROCEDURE getStudentName(
   IN sid INT,
   OUT sname VARCHAR(50)
)
BEGIN
   SELECT name INTO sname FROM student WHERE id = sid;
END;
```

- **Java Code to Call Stored Procedure**

```
CallableStatement cs =
   con.prepareCall("{call getStudentName(?, ?)}");

cs.setInt(1, 101);                    // IN parameter
cs.registerOutParameter(2, Types.VARCHAR); // OUT parameter

cs.execute();

String name = cs.getString(2);
System.out.println("Student Name: " + name);
```

Que: 3

Working with IN and OUT Parameters in Stored Procedures

Ans:

1. **IN Parameters**
   - Used to send values from Java to the stored procedure
   - Set using setXXX() methods

     ```
     cs.setInt(1, 101);
     ```
2. **OUT Parameters**
   - Used to receive values from the stored procedure
   - Must be registered using registerOutParameter()

     ```
     cs.registerOutParameter(2, Types.VARCHAR);
     ```
3. **INOUT Parameters**
   - Used for both input and output

     ```
     CREATE PROCEDURE updateMarks(
       INOUT marks INT
     )
     BEGIN
       SET marks = marks + 10;
     END;
     ```

➢ **Example:**

```
CallableStatement cs =
    con.prepareCall("{call updateMarks(?)}");

cs.setInt(1, 70);
cs.registerOutParameter(1, Types.INTEGER);

cs.execute();

int updatedMarks = cs.getInt(1);
```

- **Summary Table:**

Parameter type	**Direction**	**Java Method**
IN	Java → DB	setXXX()
OUT	DB → Java	registerOutParameter() + getXXX()
INOUT	Both	setXXX() + registerOutParameter()

- **Interview One-Line Answers**
  - **CallableStatement:** Used to call stored procedures in JDBC
  - **IN Parameter:** Sends data to stored procedure
  - **OUT Parameter:** Retrieves data from stored procedure
  - **INOUT Parameter:** Used for both input and output