1. **Introduction to JDBC**

**Q1.What is JDBC (Java Database Connectivity)?**

**ANS :-**

**JDBC (Java Database Connectivity)** is an API (Application Programming Interface) in Java that allows applications to interact with databases.

JDBC is a java API to connect and execute the query with the database.

It is a part of JavaSE (Java standard Edition).

There are four types of JDBC drivers :-

1. JDBC – ODBC Bridge driver
2. Native Driver
3. Network Protocol Driver
4. Thin driver

**Q2.Importance of JDBC in Java Programming**

**ANS :-** JDBC is essential in Java programming as it provides a **robust, secure, and efficient** way to connect Java applications with databases.

With JDBC, we can send SQL queries from Java code to store, update, delete, or get data from a database.

It is simple, secure, and works with many different types of databases.

**Q3.JDBC Architecture: Driver Manager, Driver, Connection, Statement, and ResultSet**

* **Driver Manager** loads the JDBC driver and establishes a connection with the database.
* A JDBC **driver** acts as a bridge between the Java application and the database.
* Represents a **connection** between Java and the database.
* **Statements** Executes SQL queries and updates the database.
* **Result set** stores the results of SQL queries and allows iteration through the data.

1. **JDBC Driver Types**

**Q1.Overview of JDBC Driver Types:**

* + - Type 1: JDBC-ODBC Bridge Driver
      * Uses **ODBC (Open Database Connectivity)** driver to communicate with databases.
      * The JDBC-ODBC driver uses an ODBC driver to connect to the database.
      * The JDBC-ODBC bridge driver converts JDBC methods calls into the ODBC function calls.
      * Oracle does not support the JDBC-ODBC bridge from java 8.
    - Type 2: Native-API Driver
      * Uses **database-specific native libraries** (DLL files) to interact with the database.
      * The Native-API driver uses the client side libraries of the database.
      * The driver converts JDBC method calls into native calls of the database API.
    - Type 3: Network Protocol Driver
      * Uses a **middleware server** to convert JDBC calls into database-specific protocol calls.
    - Type 4: Thin Driver
      * Also called the **"Direct-to-Database" driver**.
      * Written **entirely in Java** and directly connects to the database **without middleware**.

**Q2.Comparison and Usage of Each Driver Type**

* **JDBC-ODBC Bridge Driver**
  + - * + Working with **legacy** systems that require ODBC.
        + Not platform Independent.
        + Poor Performance
        + Low Security
        + Easy to use
        + Can be easily connected to any database.
* **Native-API Driver**
  + - * + Application needs **database-specific features**.
        + Not platform Independent
        + Moderate Performance
        + Moderate Security
        + Performance upgraded than JDBC-ODBC bridge driver
* **Network Protocol Driver**
  + - * + A **middleware server** is available to manage database connections.
        + Platform Independent
        + Moderate Performance
        + Moderate Security
* **Thin Driver**
  + - * + Using **Modern applications** (web, enterprise, cloud-based).
        + Platform Independent (pure java)
        + Excellent Performance
        + High Security

1. **Steps for Creating JDBC Connections**

**Q1.Step-by-Step Process to Establish a JDBC Connection:**

1. Import the JDBC packages
   * Import necessary JDBC classes from the java.sql package.
2. Register the JDBC driver
   * Load the JDBC driver for the specific database you are using.
   * Class.forName("com.mysql.cj.jdbc.Driver");
3. Open a connection to the database
   * Use DriverManager.getConnection() to establish a connection to the database.
   * Connection cn=DriverManager.getConnection(“jdbc:mysql://localhost:3306/java”,”root”,””);
4. Create a statement
   * A Statement object is used to execute SQL queries.
   * Three types of statements:
     + **Statement**
     + **PreparedStatement**
     + **CallableStatement**
5. Execute SQL queries
   * Use **executeQuery**() for SELECT statements (returns a **ResultSet**).
   * Use **executeUpdate**() for INSERT, UPDATE, and DELETE (returns an integer indicating affected rows).
6. Process the result set
   * Iterate through the ResultSet to fetch data.
7. Close the connection
   * Always **close** the ResultSet, Statement, and Connection.
8. **Types of JDBC Statements**

**Q1.Overview of JDBC Statements:**

**Statement:**

* The statement provides methods to execute queries with the database.
* The statement interface is a factory of resultset.
* Execute simple SQL queries without parameters.
* A Statement is used to execute **static SQL queries** that do not require input.

Class.*forname(com.mysql.jdbc.driver);*

Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM employees");

while (rs.next()) {

System.out.println("ID: " + rs.getInt("id"));

}

**Parameters :**

* **PreparedStatement:**
  + - The preparedStatement is a sub interface of Statement.
    - It is used to execute parameterized query.
    - Precompiled SQL statements for queries with parameters.
    - Instead of Statement, use PreparedStatement for **better performance and security**.
* **CallableStatement:** 
  + - * Used to call stored procedures.

1. **JDBC CRUD Operations (Insert, Update, Select, Delete)**
   * **Insert:**
   * Adding a new record to the database.
   * With executeUpdate(“SQL\_QUERY”) we insert the data.
   * **Update:**
   * Modifying existing records.
   * Same as we had insert data with executeUpdate(“SQL\_QUERY”) for updatation of data.
   * **Select:**
   * Retrieving records from the database.
   * While retrieving records from database we store the result in ResultSet.
   * ResultSet rs = stmt.executeQuery(“SQL\_QUERY”);
   * **Delete:**
   * Removing records from the database.
   * Same as we used executeUpdate(“SQL\_QUERY”) for deletion also.
2. **ResultSet Interface**

**Q1.What is ResultSet in JDBC?**

* Result Set is an object that holds the results of a SQL query executed using a Statement or Prepared Statement.
* It provides a way to access and manipulate the data returned from the database.
* The object of resultset maintains a cursor pointing to a row of a tabl.
* Initially, the cursor points to the first row.

**Q2.Navigating through ResultSet (first, last, next, previous)**

* The Result Set interface provides several methods to navigate through the result set:
* **first():** Moves the cursor to the first row in the result set.
* **last():** Moves the cursor to the last row in the result set.
* **next():** Moves the cursor to the one row next from the current position in the result set.
* **previous():** Moves the cursor to the previous row in the result set.

**Q3.Working with ResultSet to retrieve data from SQL queries**

Use a Statement or Prepared Statement to execute the SQL query.

* Get the Result Set object from the Statement or Prepared Statement.
* Use navigation methods like next(), first(), last(), and previous() to move through the result set.
* Use getter methods like get String(), getInt(), and get Date() to retrieve data from the result set.

1. **Database Metadata**

**Q1.What is DatabaseMetaData?**

ANS :- An interface in JDBC that provides information about the database.

DatabaseMetaData is an **interface in JDBC** that provides detailed information about the **database** and its capabilities.

**Q2.Importance of Database Metadata in JDBC**

ANS :- Provides crucial information about the database structure schema.

**Q3.Methods provided by DatabaseMetaData (getDatabaseProductName, getTables, etc.)**

* **getDatabaseProductName():** database product name.
* **getDriverName**(): Returns the JDBC driver name.
* **get Tables():** Returns a Result Set containing information about the database tables.
* **getCo**l**umns()**: Returns a Result Set containing information about the columns in a table.
* **getPrimaryKeys():** Returns a Result Set containing information about the primary keys in a table.

1. **ResultSet Metadata**

**Q1.What is ResultSetMetaData?**

It provides information about the structure and properties of a Result Set, such as column names, data types, and column counts.

**Q2. Methods in ResultSetMetaData (getColumnCount, getColumnName, getColumnType)**

**getColumnCount():** Returns the number of columns in the Result Set.

**getColumnName():** Returns the name of a specific column.

**getColumnType():** Returns the data type of a specific column.

1. **Practical SQL Query Examples**

**Q1.Write SQL queries for:**

* + - Inserting a record into a table.
      * INSERT INTO table name (column1, column2, column3) VALUES ('value1', 'value2', 'value3');
    - Updating specific fields of a record.
      * UPDATE table name SET column1 = 'new\_value1', column2 = 'new\_value2' WHERE condition;
    - Selecting records based on certain conditions.
      * SELECT column1, column2 FROM table name WHERE condition;
    - Deleting specific records.
      * DELETE FROM table name WHERE condition;

1. **Practical Example 1: Swing GUI for CRUD Operations**

**Q1.Introduction to Java Swing for GUI development**

ANS :- GUI tool kit for Java that provides components and tools for building desktop applications with graphical user interfaces.

**Q2.How to integrate Swing components with JDBC for CRUD operations**

* + - Create a Swing Frame
    - Add Swing Components
    - Establish a Database Connection
    - Perform CRUD Operations
    - Display Data
    - Handle Button Events