Java – RDBMS & Database Programming with JDBC

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

=>

JDBC is an API (Application Programming Interface) in Java that allows Java applications to connect and interact with databases.

It provides classes and interfaces to send SQL queries to a database and process the results.

JDBC acts as a bridge between Java code and the database driver.

**Q2) Importance of JDBC in Java Programming :-**

=>Database Communication → Allows Java apps to read/write data from databases.

Vendor Independence → Works with MySQL, Oracle, PostgreSQL, etc., without rewriting core logic (only change the driver).

Secure Data Access → Supports PreparedStatement to prevent SQL injection.

Supports CRUD → Insert, Update, Delete, and Select operations easily.

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

=>

Components:

DriverManager → Manages registered drivers and creates DB connections.

Driver → Interface implemented by database vendors (e.g., MySQL driver).

Connection → Represents an active connection to the DB.

Statement → Sends SQL queries to the DB.

ResultSet → Stores and processes query results.

Q4)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

=>

Type Name How it Works

Pros Cons

I JDBC-ODBC Bridge Translates JDBC calls to ODBC Easy to use Slow, needs ODBC installed

II Native-API Driver Uses native DB client libraries Faster than Type 1 Platform-dependent

III Network Protocol Driver Uses middleware server DB independent Requires middleware

IV Thin Driver Directly connects via DB protocol Fast, portable Needs vendor-specific driver JAR

**Q5)Comparison and Usage of Each Driver Type**

=>

Type 1 → Obsolete, rarely used now.

Type 2 → Legacy enterprise apps.

Type 3 → Multi-database environments.

Type 4 → Modern choice (e.g., mysql-connector-j).

**Q6) 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 result set

7. Close the connection

=>

// I. Import JDBC packages

import java.sql.\*;

// II. Register Driver

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

// III. Open Connection

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/testdb", "root", "password");

// IV. Create Statement

Statement stmt = con.createStatement();

// V. Execute SQL Query

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

// VI. Process Results

while(rs.next()) {

System.out.println(rs.getInt("id") + " " + rs.getString("name"));

}

// VII. Close Connection

rs.close();

stmt.close();

con.close();

**Q7) 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.

Q8) Insert: Adding a new record to the database.

-- Insert

INSERT INTO users (id, name, email) VALUES (1, 'John', 'john@example.com');

**Q9) Update: Modifying existing records.**

=>

-- Update

UPDATE users SET email='new@example.com' WHERE id=1;

**Q10)Select: Retrieving records from the database.**

=>

-- Select

SELECT \* FROM users WHERE id=1;

Q11) Delete: Removing records from the database.

=>

-- Delete

DELETE FROM users WHERE id=1;

**Q12) What is ResultSet in JDBC?**

=>

Definition:

ResultSet is an object in JDBC that holds the data returned by executing a SELECT SQL query.

Purpose:

Allows reading table data row by row and column by column.

Created by:

Statement.executeQuery() or PreparedStatement.executeQuery().

**Q13) Navigating through ResultSet (first, last, next, previous)**

=>

Common navigation methods:

rs.next() → Moves to the next row.

rs.previous() → Moves to the previous row (only in scrollable ResultSet).

rs.first() → Moves to the first row.

rs.last() → Moves to the last row.

rs.absolute(int row) → Moves to a specific row number.

Note:

Default ResultSet is forward-only; to navigate backward, use:

java

Copy

Edit

Statement stmt = con.createStatement(

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY

);

**Q14) Working with ResultSet to retrieve data from SQL queries**

=>

Example:

java

Copy

Edit

while(rs.next()) {

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

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

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

}

**Q 15) What is DatabaseMetaData?**

=>

Definition:

An interface in JDBC providing information about the database and driver.

Purpose:

Used to get DB details without writing queries.

Q16) Importance of Database Metadata in JDBC

=>

Useful for:

Checking DB version.

Listing tables, columns, and supported features.

Building dynamic applications that work with multiple DBs.

**Q17)Methods provided by DatabaseMetaData (getDatabaseProductName, getTables,**

**etc.)**

**=>**

getDatabaseProductName() → Returns DB name.

getDatabaseProductVersion() → Returns DB version.

getDriverName() → Returns JDBC driver name.

getTables() → Lists tables in the DB.

supportsTransactions() → Checks if transactions are supported.

**Q18) What is ResultSetMetaData?**

**=>**

Definition:

Interface that gives information about the columns in a ResultSet.

Purpose:

Allows analyzing the structure of query results.

**Q19) Importance of ResultSet Metadata in analyzing the structure of query results**

=>

Writing generic programs that can handle any query result.

Displaying table data dynamically in GUIs.

Debugging SQL results.

Q20) Methods in ResultSetMetaData (getColumnCount, getColumnName,

getColumnType)

=>

getColumnCount() → Number of columns.

getColumnName(int column) → Name of a column.

getColumnType(int column) → SQL type of a column.

getColumnTypeName(int column) → Name of SQL type.

**Q21) Write SQL queries for:**

Inserting a record into a table.

Updating specific fields of a record.

Selecting records based on certain conditions.

Deleting specific records.

=>

-- Insert

INSERT INTO users (id, name, email) VALUES (1, 'John', 'john@example.com');

-- Update

UPDATE users SET email = 'new@example.com' WHERE id = 1;

-- Select

SELECT \* FROM users WHERE id = 1;

-- Delete

DELETE FROM users WHERE id = 1;

**Q22)Implement these queries in Java using JDBC.**

=>

// Insert

PreparedStatement ps = con.prepareStatement("INSERT INTO users VALUES(?, ?, ?)");

ps.setInt(1, 1);

ps.setString(2, "John");

ps.setString(3, "john@example.com");

ps.executeUpdate();

// Update

PreparedStatement ps2 = con.prepareStatement("UPDATE users SET email=? WHERE id=?");

ps2.setString(1, "new@example.com");

ps2.setInt(2, 1);

ps2.executeUpdate();

// Select

PreparedStatement ps3 = con.prepareStatement("SELECT \* FROM users WHERE id=?");

ps3.setInt(1, 1);

ResultSet rs = ps3.executeQuery();

while(rs.next()) {

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

}

// Delete

PreparedStatement ps4 = con.prepareStatement("DELETE FROM users WHERE id=?");

ps4.setInt(1, 1);

ps4.executeUpdate();

Q23) Introduction to Java Swing for GUI development

=>

Java Swing is a GUI toolkit for creating desktop applications.

Features:

Components: JFrame, JButton, JTextField, etc.

Platform-independent.

Event-driven programming.

**Q24) How to integrate Swing components with JDBC for CRUD operations**

=>

Steps:

Design form in Swing with input fields and buttons.

Add event listeners to buttons.

In listeners, write JDBC code to perform CRUD.

Example:

java

Copy

Edit

insertBtn.addActionListener(e -> {

PreparedStatement ps = con.prepareStatement("INSERT INTO users VALUES(?, ?, ?)");

ps.setInt(1, Integer.parseInt(idField.getText()));

ps.setString(2, nameField.getText());

ps.setString(3, emailField.getText());

ps.executeUpdate();

});

**Q25) What is a CallableStatement?**

=>

Definition:

A JDBC interface to execute stored procedures in the database.

Created by:

Connection.prepareCall() method.

**Q26) How to call stored procedures using CallableStatement in JDBC**

=>

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

cs.setInt(1, 1);

ResultSet rs = cs.executeQuery();

Q27) Working with IN and OUT parameters in stored procedures

=>

CREATE PROCEDURE getUser(IN uid INT, OUT uname VARCHAR(50))

BEGIN

SELECT name INTO uname FROM users WHERE id = uid;

END;

Java Code:

java

Copy

Edit

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

cs.setInt(1, 1); // IN param

cs.registerOutParameter(2, Types.VARCHAR); // OUT param

cs.execute();

System.out.println("User Name: " + cs.getString(2));