Module 7 – Java – RDBMS & Database Programming with JDBC

1. Introduction to JDBC

What is JDBC (Java Database Connectivity)?

JDBC is a **Java API** that allows Java programs to **connect and interact with databases** such as MySQL, Oracle, PostgreSQL, etc. It provides standard classes and interfaces to execute SQL queries from Java.

Example:
Using JDBC, a Java program can fetch student records stored in a MySQL database.

Importance of JDBC in Java Programming

- Enables **database-driven applications**
- Platform-independent database access
- Supports **CRUD operations**
- Used in **enterprise, web, and desktop applications**

Example:
Online banking systems use JDBC to store and retrieve account details.

JDBC Architecture

JDBC follows a **layered architecture**:

1. **DriverManager** – Manages database drivers
2. **Driver** – Communicates with the database
3. **Connection** – Represents database connection
4. **Statement** – Executes SQL queries
5. **ResultSet** – Holds query results

Example Flow:
Java App → DriverManager → JDBC Driver → Database

2. JDBC Driver Types

Overview of JDBC Driver Types

Type 1: JDBC-ODBC Bridge Driver

- Converts JDBC calls into ODBC calls
- Requires ODBC installation
- **Slow and outdated**

Example:
Used in early Java versions for MS Access databases.

Type 2: Native-API Driver

- Uses database-specific native libraries
- Faster than Type 1
- Platform dependent

Example:
Oracle OCI driver.

Type 3: Network Protocol Driver

- Uses middleware server
- Database-independent
- Suitable for enterprise applications

Example:
Java application → middleware → database.

Type 4: Thin Driver

- Pure Java driver
- Direct communication with database

- **Most popular and recommended**

Example:
MySQL Connector/J.

Comparison and Usage

Driver Type	Performance	Platform	Usage
Type 1	Low	Dependent	Legacy
Type 2	Medium	Dependent	Limited
Type 3	High	Independent	Enterprise
Type 4	Very High	Independent	Most Used

3. Steps for Creating JDBC Connections

Step-by-Step Process

1. Import JDBC packages
2. Register the JDBC driver
3. Open database connection
4. Create Statement
5. Execute SQL query
6. Process ResultSet
7. Close connection

Example:
A Java program connects to MySQL, executes a SELECT query, and prints results.

Best JDBC Driver for MySQL

- **Type 4 (Thin Driver)** is best

- Reason: Fast, secure, platform-independent

Example:
MySQL Connector/J is widely used in Java applications.

4. Types of JDBC Statements

Statement

- Executes simple SQL queries
- No parameters
- Less secure

Example:
Fetching all records from a table.

PreparedStatement

- Precompiled SQL
- Supports parameters
- Prevents SQL injection
- Faster execution

Example:
Selecting user details using user ID.

CallableStatement

- Used to call stored procedures
- Supports IN, OUT parameters

Example:
Calling a procedure to calculate employee salary.

Differences Between Statements

Feature	Statement	PreparedStatement	CallableStatement
Parameters	No	Yes	Yes
Performance	Low	High	High
Security	Low	High	High
Stored Procedure	No	No	Yes

5. JDBC CRUD Operations

Insert Operation

Adds new data into the database.

Example:
Adding a new student record.

Update Operation

Modifies existing records.

Example:
Updating student email ID.

Select Operation

Retrieves data from database.

Example:
Fetching all employee records.

Delete Operation

Removes records from database.

Example:
Deleting inactive users.

6. ResultSet Interface

What is ResultSet?

ResultSet stores the **data returned by SELECT queries**.

Navigating ResultSet

- next() – Move to next row
- previous() – Move to previous row
- first() – First row
- last() – Last row

Example:
Looping through records and printing student names.

Retrieving Data from ResultSet

- getInt()
- getString()
- getDouble()

Example:
Fetching ID and name from result set.

7. DatabaseMetaData

What is DatabaseMetaData?

Provides information about the **database itself**, such as name, version, tables, and features.

Importance of DatabaseMetaData

- Helps understand database structure
- Useful for dynamic applications
- Supports database portability

Common DatabaseMetaData Methods

- getDatabaseProductName()
- getDatabaseProductVersion()

- getTables()
- supportsTransactions()

Example:
Displaying database name and version at runtime.

8. ResultSetMetaData

What is ResultSetMetaData?

Provides information about the **columns in a ResultSet**.

Importance of ResultSetMetaData

- Helps analyze query structure
- Useful for dynamic table display
- No prior column knowledge required

Common ResultSetMetaData Methods

- getColumnCount()
- getColumnName()
- getColumnType()

Example:
Displaying column names and data types of a table.

9. Practical SQL Query Examples (Theory)

Insert Query

Adds a record.

Example:
Insert student details into table.

Update Query

Updates specific fields.

Example:
Change email ID based on student ID.

Select Query

Fetches records with conditions.

Example:
Select employees with salary > 50,000.

Delete Query

Deletes specific records.

Example:
Delete records where status is inactive.

10. Swing GUI with JDBC (Theory Only)

Introduction to Java Swing

Swing is a **GUI toolkit** for creating desktop applications in Java.

Integrating Swing with JDBC

- Swing handles UI
- JDBC handles database
- Buttons trigger database operations

Example:
Clicking "Insert" button stores data in MySQL.

11. CallableStatement with IN and OUT Parameters

What is CallableStatement?

Used to execute **stored procedures** in databases.

IN and OUT Parameters

- **IN**: Sends input to procedure
- **OUT**: Receives output from procedure

Usage of CallableStatement

- Improves performance
- Encapsulates business logic in database

Example:

Passing employee ID (IN) and receiving full name (OUT).