
🌟 FULL PRACTICAL ANSWER (EXTENDED VERSION)

🎯 AIM

To write a Java program using JDBC (Java Database Connectivity) to connect with a MySQL/Oracle database and perform database navigation operations such as:

- Adding Records
- Viewing Records
- Updating Records
- Deleting Records

This program demonstrates front-end to database connectivity using Java as the front end and MySQL as the back end.

📘 THEORY (EXPANDED — EXAM MAXIMUM MARKS VERSION)

🔹 1. Introduction to Database Connectivity

A database is used to store, update, delete, and retrieve structured information. Front-end applications like Java need a way to connect and communicate with databases such as MySQL or Oracle.

For this purpose, Java provides a powerful API called JDBC — Java Database Connectivity.

🔹 2. What is JDBC?

JDBC stands for Java Database Connectivity, a standard Java API that enables Java programs to interact with relational databases using SQL.

JDBC allows Java applications to:

- Establish a connection with a database
- Send SQL statements
- Retrieve data from ResultSet
- Perform CRUD (Create, Read, Update, Delete) operations

🔹 3. JDBC Architecture

JDBC uses the following architecture:

Application Layer

Java Program (Front end)

JDBC Driver Manager

Responsible for loading database drivers and establishing connection.

JDBC Driver

Vendor-specific driver such as:

- com.mysql.cj.jdbc.Driver (MySQL)
- oracle.jdbc.driver.OracleDriver (Oracle)

Database

Relational database (MySQL/Oracle)

🔹 4. JDBC Steps (Very Important for Viva)

A JDBC program generally follows 7 steps:

Step 1: Load the JDBC Driver

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

Step 2: Establish the Connection

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

Step 3: Create Statement/PreparedStatement

- Statement → for static SQL
- PreparedStatement → for dynamic SQL

Step 4: Execute SQL Query

- SELECT → executeQuery()
- INSERT/UPDATE/DELETE → executeUpdate()

Step 5: Process ResultSet

Used only for SELECT queries.

Step 6: Close ResultSet/Statement

After processing.

Step 7: Close Connection

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

🔹 5. PreparedStatement

PreparedStatement is preferred over Statement because:

- **It prevents SQL injection**
- **It allows setting parameters**
- **It improves performance**

Example:

```
PreparedStatement pst = con.prepareStatement(
  "insert into emp values(?, ?, ?)"
);
```

🔹 6. CRUD / Navigation Operations

CRUD stands for:

- **Create (Insert)**
- **Read (Select)**
- **Update**
- **Delete**

Navigation means moving through records and performing actions like:

- **Display all**
- **Modify selected data**
- **Delete selected row**

🧾 PROGRAM (FULL, CLEAN, EXTENDED VERSION)

Java Program to Connect MySQL and Perform CRUD Operations

```
package mydb;

import java.sql.*;
import java.util.Scanner;

public class SimpleDBExample {

  public static void main(String[] args) {

    String url = "jdbc:mysql://localhost:3306/testdb";
    String user = "root";
    String pass = "root";

    try (Scanner sc = new Scanner(System.in)) {

      try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(url, user, pass);
        System.out.println("Connected to Database Successfully!");

        while (true) {

          System.out.println("\n========== MENU ==========");
          System.out.println("1. Add Employee");
          System.out.println("2. View Employees");
          System.out.println("3. Update Employee");
          System.out.println("4. Delete Employee");
          System.out.println("5. Exit");
System.out.print("Enter your choice: ");

int ch = sc.nextInt();

switch (ch) {

  // ---------- ADD EMPLOYEE ----------
  case 1:
    System.out.print("Enter Employee ID: ");
    int id = sc.nextInt();
    System.out.print("Enter Name: ");
    String name = sc.next();
    System.out.print("Enter City: ");
    String city = sc.next();

    PreparedStatement pst = con.prepareStatement(
      "insert into emp(eid, ename, city) values(?, ?, ?)"
    );
    pst.setInt(1, id);
    pst.setString(2, name);
    pst.setString(3, city);
    pst.executeUpdate();

    System.out.println("Record Added Successfully!");
    break;

  // ---------- VIEW EMPLOYEES ----------
  case 2:
```

```
  Statement st = con.createStatement();
  ResultSet rs = st.executeQuery("select * from emp");

  System.out.println("\nEid\tName\tCity");
  System.out.println("---------------------------");

  while (rs.next()) {
    System.out.println(rs.getInt("eid") + "\t"
        + rs.getString("ename") + "\t"
        + rs.getString("city"));
  }
  break;

// ---------- UPDATE EMPLOYEE ----------
case 3:
  System.out.print("Enter Employee ID to update: ");
  int uid = sc.nextInt();

  System.out.print("Enter new Name: ");
  String newName = sc.next();

  System.out.print("Enter new City: ");
  String newCity = sc.next();

  PreparedStatement up = con.prepareStatement(
    "update emp set ename=?, city=? where eid=?"
  );
  up.setString(1, newName);
  up.setString(2, newCity);
  up.setInt(3, uid);

  int rowsUpdated = up.executeUpdate();

  if (rowsUpdated > 0)
    System.out.println("Record Updated Successfully!");
  else
    System.out.println("Employee Not Found!");

  break;

// ---------- DELETE EMPLOYEE ----------
case 4:
  System.out.print("Enter Employee ID to delete: ");
  int did = sc.nextInt();

  PreparedStatement del = con.prepareStatement(
    "delete from emp where eid=?"
  );
  del.setInt(1, did);

  int rowsDeleted = del.executeUpdate();

  if (rowsDeleted > 0)
    System.out.println("Record Deleted Successfully!");
  else
    System.out.println("Employee Not Found!");
```

```
                break;

              // ---------- EXIT ----------
              case 5:
                con.close();
                System.out.println("Database Disconnected. Exiting Program...");
                System.exit(0);

              default:
                System.out.println("Invalid Choice! Try Again.");
            }
          }

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

📘 FULL DETAILED EXPLANATION (BIG VERSION)

✔ 1. Loading MySQL JDBC Driver

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

This loads MySQL Driver into memory so Java can communicate with MySQL.

✔ 2. Establishing the Connection

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

- **url → database address**
- **user → MySQL username**
- **pass → MySQL password**
  Creates a link between Java program and MySQL.

✔ 3. Menu-Driven Interface

The menu repeats using an infinite loop:

1. Add Employee

2. View Employees

3. Update Employee

4. Delete Employee

5. Exit

User selects operations.

✔ 4. Insert Operation (Add)

insert into emp values(?, ?, ?)

Uses PreparedStatement for safety.

✔ 5. Select Operation (View)

ResultSet rs = st.executeQuery("select * from emp");

Fetches all rows and displays them.

✔ 6. Update Operation

update emp set ename=?, city=? where eid=?

Updates name & city for selected employee.

✔ 7. Delete Operation

```
delete from emp where eid=?
```

Deletes specific row based on employee ID.

✔ 8. Exit Program

```
System.exit(0);
```

Closes the connection and stops the program.

📊 SAMPLE OUTPUT

```
Connected to Database Successfully!


========== MENU ==========

1. Add Employee

2. View Employees

3. Update Employee

4. Delete Employee

5. Exit

Enter choice: 1


Enter Employee ID: 101

Enter Name: Ajay

Enter City: Nashik

Record Added Successfully!
```

🟩 ADVANTAGES OF JDBC

✔ Platform-independent (Java-based)
✔ Secure (PreparedStatement prevents SQL Injection)
✔ Fast and efficient
✔ Supports all relational databases
✔ Easy to integrate with GUI apps like Swing/JavaFX

🟥 LIMITATIONS OF JDBC

❌ Requires driver installation
❌ More code than ORM frameworks
❌ Manual exception handling needed

🎤 VIVA QUESTIONS & ANSWERS (EXTENDED VERSION)

Q1. What is JDBC?

JDBC (Java Database Connectivity) is an API that allows Java applications to interact with relational databases using SQL queries.

Q2. Why do we use Class.forName()?

To load the JDBC driver dynamically so the DriverManager can establish the connection.

Q3. What is PreparedStatement?

A precompiled SQL statement that allows parameter binding using '?' and prevents SQL Injection.

Q4. Difference between Statement and PreparedStatement?

Statement	PreparedStatement
Static SQL	Dynamic SQL
No parameters	Supports '?'
Less secure	More secure
Slower	Faster

Q5. What is ResultSet?

ResultSet is a table-like object returned by SELECT queries allowing row-wise read access.

Q6. What are CRUD operations?

Create (Insert), Read (Select), Update, Delete.

Q7. Can the same code work for Oracle?

Yes. Only the driver and URL must be changed:

Class.forName("oracle.jdbc.driver.OracleDriver");

jdbc:oracle:thin:@localhost:1521:xe

Q8. What is DriverManager?

A class responsible for managing JDBC drivers and creating database connections.

🧾 EXTRA QUESTIONS (in case external asks more)

✓ Difference between executeQuery() and executeUpdate()
✓ What is SQL Injection?
✓ What happens if connection is not closed?
✓ What is metadata in JDBC?
✓ What is autocommit?

🟦 CONCLUSION

This program demonstrates how Java connects to a database using JDBC and performs navigation operations like addition, retrieval, updation, and deletion of employee data. It implements secure and structured database access using PreparedStatement and ResultSet.
