Module 2 – Java – RDBMS & Database Programming with JDBC

➢ Introduction to JDBC

• Theory:

1. What is JDBC (Java Database Connectivity)?

Ans:

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

2. Importance of JDBC in Java Programming

Ans:

Following are the importance of jdbc in java programming :-

(i) Database Connectivity:

JDBC allows Java applications to connect to various databases like MySQL, Oracle, PostgreSQL, etc., and perform operations such as insert, update, delete, and retrieve data.

(ii) Platform Independence:

Since Java is platform-independent, JDBC also supports database access across different platforms, making applications portable and flexible.

(iii) Supports Multiple Databases:

With JDBC, you can work with **any relational database** by just changing the driver and connection string. No need to change your core logic.

(iv) Transaction Management:

JDBC provides support for transaction control (commit, rollback), which is important in applications that require data integrity and consistency.

(v) Bridge Between Java & SQL:

JDBC acts as a **bridge** that allows Java to communicate with SQL databases. Without JDBC, it would be difficult to access or manage databases in Java.

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

Ans:

Driver Manager:

It manages a list of database drivers. It selects the correct driver from the list to connect to a database

You use DriverManager.getConnection()

to get the connection object.

Ex: Connection con = DriverManager.getConnection(url, user, password);

Driver:

It is a software component (like .jar file) that translates Java JDBC calls into database-specific calls.

Ex drivers:

MySQL- com.mysql.cj.jdbc.Driver

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

Connection:

Represents a connection between Java and the database.It is used to send SQL queries and manage transactions.

Connection con = DriverManager.getConnection(...);

Statement:

Statement / PreparedStatement

Used to execute SQL queries.

Statement: It is mostly used for select query.

PreparedStatement: it is used insert , update ,delete query or also For dynamic or parameterized queries (safer).

Ex:

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

ResultSet:

It holds the result of a query (data retrieved from the database).You can use .next(), .getString(), .getInt() to access data.

Ex:

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

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

- **Lab Exercise:**

4. Write a simple Java program to connect to a MySQL database using JDBC.

Ans:

```
    try {

    Class.forName("com.mysql.cj.jdbc.Driver");
                    } catch (ClassNotFoundException
e) {
                        System.out.println("Driver
class not found");
                    }
```

5. Demonstrate the process of loading a JDBC driver and establishing a connection.

Ans:

```
    Connection cn=null;
                    try {
```

```
cn=DriverManager.getConnection("jdbc:mysql:
//localhost/mahek", "root", "");
                    } catch (SQLException e) {

System.out.println("connction not found");
```

- **JDBC Driver Types**
  - **Theory:**

6.Overview of JDBC Driver Types:

♣ Type 1: JDBC-ODBC Bridge Driver :

A driver that uses ODBC (Open Database Connectivity) to connect Java applications to the database. It acts as a bridge between JDBC and ODBC.

♣ Type 2: Native-API Driver:

A driver that converts JDBC calls into the native database API using C/C++ libraries.
It needs database-specific native code installed on the machine.

♣ Type 3: Network Protocol Driver:

A driver that sends JDBC calls to a middleware server, which then communicates with the database. the driver is platform-independent, but needs an intermediate server.

♣ Type 4: Thin Driver:

A driver written completely in Java that converts JDBC calls directly into the database-specific protocol. It doesn't need any native libraries or middleware.

- **Lab Exercise:**

7. Identify which driver your Java program uses to connect to MySQL.

Ans:

Thin driver java program uses to connect to MySQL.

➢ **Steps for Creating JDBC Connections**

- **Theory:**

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

Step-1: Import the JDBC packages:

You need to import the required **JDBC classes** from the java.sql package.

import java.sql.*;

Step-2: Register the JDBC driver:
This step loads the **JDBC driver class** into memory.
Class.forName("com.mysql.cj.jdbc.Driver");

Step-3: Open a connection to the database:
Use DriverManager.getConnection() to establish a connection.
Connection con =
DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mahek",
"username", " ");

Step-4: Create a statement:
Create a Statement or PreparedStatement object to execute SQL commands.
Statement stmt = con.createStatement();
Or
PreparedStatement pstmt =
con.prepareStatement("SELECT * FROM users WHERE id = ?");

Step-5: Execute SQL queries:

Execute your SQL command using executeQuery() (for SELECT) or executeUpdate() (for INSERT/UPDATE/DELETE).

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

Step-6: Process the result set:

Use a loop to read the data returned from the database.

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

Step-7: Close the connection:

Always close ResultSet, Statement, and Connection to release resources.

```
rs.close();
stmt.close();
con.close();
```

- Lab Exercise:

9. Write a Java program to establish a connection to a database and print a confirmation message upon successful connection.

Ans:

```
    import java.sql.*;


public class connectivity {

 public static void main(String[] args) {

                    //Driver class register
                    try {

 Class.forName("com.mysql.cj.jdbc.Driver");
                    } catch (ClassNotFoundException
e) {
                         System.out.println("Driver
class not found");
                    }
                    Connection cn=null;
                    try {

 cn=DriverManager.getConnection("jdbc:mysql:
//localhost/mahek", "root", "");

 System.out.println("connection
successfully...");
                    } catch (SQLException e) {
```

```
System.out.println("connction not found");
                    }

                    }
}
```

- **Types of JDBC Statements**
  - **Theory:**

10.Overview of JDBC Statements

Ans:

Statement: Executes simple SQL queries without parameters.

PreparedStatement: Precompiled SQL statements for queries withparameters.

CallableStatement: Used to call stored procedures.

11.Differences between Statement, PreparedStatement, and CallableStatement

Ans:

St at e m e n t	Prepared statement	
To run normal SQL queries	To run SQL queries with parameters	To run stored procedures in DB
It is St at ic	It is Dynamic	It is Procedure Call

It is Not safe	It is safe	It is safe
It is slower, compiled every time	It is faster, query is precompiled	It is faster, stored procedure is already compiled
Mostly Simple queries like SELECT	Login forms, insert/update/delete	Complex business logic stored in DB

- **Lab Exercise:**

12.Create a program that inserts, updates, selects, and deletes data using Statement.

Ans:

```
import java.sql.*;

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


        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
```

```
        try {
            // Load Driver

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

            // Connect to DB
            con =
DriverManager.getConnection("jdbc:mysql://lo
calhost/try", "root", "");
            System.out.println("Connected to
database.");

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

        } catch (Exception e) {
            System.out.println("Connection Error:");
            e.printStackTrace();
        }

        // INSERT Operation
        try {
            String insert = "INSERT INTO students
(id, name, age) VALUES (1, 'John', 22)";
            int result = stmt.executeUpdate(insert);
```

```
            System.out.println("Inserted: " + result +
" row(s).");
        } catch (SQLException e) {
            System.out.println("Insert Error:");
            e.printStackTrace();
        }

        // UPDATE Operation
        try {
            String update = "UPDATE students SET
age = 23 WHERE id = 1";
            int result = stmt.executeUpdate(update);
            System.out.println("Updated: " + result
+ " row(s).");
        } catch (SQLException e) {
            System.out.println("Update Error:");
            e.printStackTrace();
        }

        // SELECT Operation
        try {
            String select = "SELECT * FROM
students";
            rs = stmt.executeQuery(select);
            System.out.println("Student Records:");
            while (rs.next()) {
```

```
                System.out.println(
                    rs.getInt("id") + " - " +
                    rs.getString("name") + " - " +
                    rs.getInt("age"));
            }
        } catch (SQLException e) {
            System.out.println("Select Error:");
            e.printStackTrace();
        }

        // DELETE Operation
        try {
            String delete = "DELETE FROM students
WHERE id = 1";
            int result = stmt.executeUpdate(delete);
            System.out.println("Deleted: " + result +
" row(s).");
        } catch (SQLException e) {
            System.out.println("Delete Error:");

        }



    }
}
```

13.Modify the program to use PreparedStatement for parameterized queries.

Ans:

```
package jdbcop;


import java.sql.*;


public class connectivity {

 public static void main(String[] args) {

                    //Driver class register
                    try {

 Class.forName("com.mysql.cj.jdbc.Driver");
                    } catch (ClassNotFoundException
e) {
                          System.out.println("Driver
class not found");
                    }
                    Connection cn=null;
```

```
                    try {

 cn=DriverManager.getConnection("jdbc:mysql:
//localhost/mahek", "root", "");

 System.out.println("connection
successfully...");
                    } catch (SQLException e) {


 System.out.println("connction not found");
                    }

                    String  inserdata="insert into
student(sname,semail,spass) values (?,?,?)";

                    try {
                        PreparedStatement
ps=cn.prepareStatement(inserdata);
                        ps.setString(1,"java");

 ps.setString(2,"java@gamil.com");
                        ps.setString(3, "jj");
                        int c=ps.executeUpdate();
                        System.out.println(c+"Insert
Record");
```

```
                } catch (SQLException e) {
                    System.out.println("Not
Inserted");
                }


                String  deletedata="delete from
student where sid=?";

                try {
                    PreparedStatement
ps=cn.prepareStatement(deletedata);
                    ps.setInt(1, 1);
                    int c=ps.executeUpdate();

 System.out.println(c+"Delete Record");
                } catch (SQLException e) {
                    System.out.println("Not
Deleted");
                }

                String  updatedata="update
student set sname=?,semail=?,spass=? where
sid=?";

                try {
```

```
                    PreparedStatement
ps=cn.prepareStatement(updatedata);
                    ps.setString(1,"react");

 ps.setString(2,"react@gmail.com");
                    ps.setString(3,"react");
                    ps.setInt(4, 2);
                    int c=ps.executeUpdate();

 System.out.println(c+"Update Record");
                } catch (SQLException e) {
                    System.out.println("Not
update");
                }

                String selectdata="select *from
student";

                try {

                    Statement
s=cn.createStatement();
                    ResultSet
rs=s.executeQuery(selectdata);
                    while(rs.next()) {
```

```
 System.out.println(rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3)+"
"+rs.getString(4));
                        }


                    } catch (Exception e) {
                        System.out.println("Not data
found");
                    }

                    }

 }
```

- **JDBC CRUD Operations (Insert, Update, Select, Delete)**

  • **Theory:**

  14.Insert: Adding a new record to the database

  Ans:

  It is Used to **add new records** into a table.

  ex: INSERT INTO table_name (...)
  VALUES (...);

  15. Update: Modifying existing records.

Ans:

It is used to **modify existing records** in a table.

ex: UPDATE table_name SET column = value WHERE condition;

16. Select: Retrieving records from the database.

Ans:

It is used to **retrieve records** from one or more tables.

ex: SELECT * FROM table_name;

17. Delete: Removing records from the database.

Ans:

It is used to **remove records** from a table.

ex: DELETE FROM table_name WHERE condition;

● **Lab Exercise:**

18. Write a Java program that performs the following CRUD operations: ♣ Insert a new record. ♣ Update an existing record. ♣ Select and display records. ♣ Delete a record.

Ans:

```
package pkgtry;

import java.sql.*;

public class three
{
   Connection con;

   public three()
   {
      try {

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

con=DriverManager.getConnection("jdbc:mysql
://localhost:3306/jtb","root","");
      } catch (Exception e) {
         System.out.println(e);
      }
   }

      public int insert(int r,String n )
      {
         int ans=0;
```

```
            try {
                PreparedStatement
ps=con.prepareStatement("insert into ntb
values(?,?)");
                ps.setInt(1, r);
                ps.setString(2, n);
                ans= ps.executeUpdate();
                return ans;
            } catch (Exception ex) {
                System.out.println(ex);
            }
        return 0;

        }

        public int delete(int r)
        {
            int ans=0;
            try {
                PreparedStatement
ps=con.prepareStatement("delete from ntb
where rno=?");
                ps.setInt(1, r);
                 ans=  ps.executeUpdate();
                 return ans;
```

```
            } catch (Exception e) {
                System.out.println(e);
            }
        return 0;
        }

      public int update(int r,String n )
        {
            int ans=0;
            try {
                PreparedStatement
ps=con.prepareStatement("update ntb set
name=? where rno=?");
                ps.setString(1, n);
                ps.setInt(2, r);
                ans= ps.executeUpdate();
                return ans;
            } catch (Exception ex) {
                System.out.println(ex);
            }
        return 0;

        }

}
```

- **ResultSet Interface**
  - **Theory:**

19.What is ResultSet in JDBC?

Ans:

A ResultSet object represents a **table of data** generated by executing a SQL query using a Statement object.

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

Ans:

First: moves the cursor to the first row

Last: moves the cursor to the last row

Next: moves the cursor to the next row

Previous: moves the cursor to te previous row

21. Working with ResultSet to retrieve data from SQL queries

Ans:

Repeated

- **Lab Exercise**:

22. Write a program that executes a SELECT query and processes the ResultSet to display records from the database.

Ans:

Repeated

- **Lab Assignment 1: Simple JDBC Program**

23. Write a Java program that connects to a MySQL database and executes a simple query to retrieve all records from a table.

Ans:

```
<table border="1" align="center"  >
  <tr><th colspan="8">Registrated Data</th></tr>
  <tr>
    <th> Id</th>
    <th>Name</th>
    <th> Email</th>
    <th>Password</th>
  </tr>

  <%
    try {
      int id=Integer.parseInt(request.getParameter("id"));
       crud obj=new crud();
       Connection conn=obj.conncet();
      PreparedStatement pst = conn.prepareStatement("select *from registration where id="+id);
```

```
        ResultSet rs = pst.executeQuery();
        while (rs.next()) {
%>
<tr><td>
        <%= rs.getInt(1)%>
    </td>
    <td>
        <%= rs.getString(2)%>
    </td>
    <td>
        <%= rs.getString(3)%>
    </td>
    <td>
        <%= rs.getString(4)%>
    </td>
</tr>


<%
        }

    } catch (Exception e) {
        out.println(e);
    }
%>
```

</table>