# Advanced Java:-

## ★ Basics of Oracle JDBC:-

Oracle JDBC is a part of the Oracle database drives used to connect JAVA applications to Oracle databases. The JDBC API provides a standard interface for Java application to interact with databases and oracle JDBC drivers facilates this connection.

### Oracle JDBC Driver Types:-

(1) Type 1 (JDBC-ODBC Bridge driver)
(2) Native-API driver [Type 2]
(3) Type-3 (Network protocol driver)
(4) Type-4 (Thin-driver)

→ Mostly Type-4 drives commanly used.

It's JDBC URL Formate: jdbc:oracle:thin:@//[hostname]:[port]/[service_name]

o Example:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

class OracleExample{
  public static void main(String args[]){
  try{
      Class.forName("oracle.jdbc.drivers.OracleDriver");
    Connection conn = DriverManager.getConnection("jdbc:oracle:
                        thin:@localhost:1521:xe","username","password");
```

```
System.out.println("Oracle Connected!");
  conn.close();
}

Catch (Exception e){
    e.printStackTrace(); }

}}
```

## ★ Program to Connect to Database [Mysql]

To connect to Mysql database using JDBC, we use MySQL JDBC driver and writing a java program that perform the connection.

Code:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException; //can be removed if wanted

class DatabaseConnection{
  psvm (string args[]){

    try{

      Connection conn = DriverManager.getConnection ("jdBc:mysql://
                                    localHost:3306/student", "root", "1234");
    System.out.println("MySQL connected!");
       conn.close();
    }
```

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

★ Loading the Drives Using MySQL:-

The MySQL JDBC drives must be loaded into java application so that the DrivesManager can use it to establish a connection.

★ Establish the connection [Mysql]:-

To establish a connection to the MYSQL database, we use the DrivesManager.getConnection method, which requires the JDBC URL, Username and password.

Code (for Loading the drives):-

```
import java.sql.DrivesManager;

Class Loadingdrives{
    ps vm (String args[]){

    try{
        Class.forName ("com.mysql.cj.jdbc.Drives");
        System.out.println("MySQL drives loaded!");
    }
    catch (Exception e){
        System.out.println(e);}
}}
```

Code (for Establishing connection):-

```
import java.sql.Connection;
import java.sql.DriverManager;

Class MySQLconnection{
 psvm(String args[]){

 try {

   Connection con = DriverManager.getConnected("jdbc:mysql://localhost:
                                    3306/student", "root", "1234");
 System.out.println("Connection Established");
     con.close();
 }

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

 }}
```

★ Creating Statement Interface and ResultSet with Stament

Interface:- (MySql)

- Statement interface in JDBC is used to execute SQL queries against the database.
- The ResultSet interface represents the result set of a query. You use it to retrieve the data returned by a query executed using a Statement.

Code :- (Statement and ResultSet interface)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

Class Statement_ResultSet{
  psvm (String[] args){

  try{
     Connection con = DriverManager.getConnection("jdbc:mysql://localhost
                      :3306/student", "root", "1234");

  //Statement

  Statement stmt= Con.createStatement();

  //ResultSet

  ResultSet rs= stmt.executeQuery("Select * from info");

  while(rs.next()){
     System.out.println("ID:"+rs.getInt("id")+", Name:"+rs.
                        getString("name")); }

     rs.close();
     stmt.close();
     Con.close(); }
catch(Exception e){
        System.out.println(e); }
}}
```

★ Prepared Statement Interface: CRUD Operation :-

Prepared statement interface in Java is used to execute SQL queries with input parameters. It's a part of the JDBC API and is more secure and efficient than the regular Statement interface because it precompiles SQL queries and helps prevent SQL injection attacks.

[SQL injection attack → It's a code injection technique in which hacker try to insert sql query in form and try to retrive data from database of a company]

C → Create
R → Read
U → Update
D → Delete

1) Create (Insert Operation) :-

The insert operation allows us to add new data to the database.

Code:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

class Create{
    psvm(String args[]){

    try{
        Connection conn = DriverManager.getConnection("jdbc:mysql://
                          localhost:3306/student", "root", "1234");

        String insert_data = "Insert into info (id, name) values (?,?)";
```

```
//Prepare Statement

PreparedStatement pstmt = conn.prepareStatement (insert-data);

//Set the parameters
pstmt.setInt (1, 902);
pstmt.setString (2, "Aditya das");

//Execute the update

int rowsInserted = pstmt.executeUpdate();

if (rowsInserted > 0) {
        System.out.println ("A new student was inserted successfully");
}}

Catch (Exception e) {
    System.out.println (e); }

}}
```

## 2) Read (Select operation):-

The select operation is used to retrieve data from the database.

Code:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
Class Read {
ps vm (String args[]) {

try {

Connection conn = DriverManager.getConnection ("jdbc:mysql://localhost
                  :3306/student", "root", "1234");

String Select_data = "Select * from info where id = 1";

// Prepared statement and its parameter

PreparedStatement pstmt = conn.prepareStatement (Select_data);
pstmt.setInt (1, 1);

// Execute the query and get result set

ResultSet rs = pstmt.executeQuery ();

// result set

While (rs.next ()){
  int id = rs.getInt ("id");
  String name = rs.getString ("name");
  System.out.println ("ID:" + id + ", Name:" + name);
  }
}

Catch (Exception e){
    System.out.println (e); }
}}
```

## (3) Update (Update operation):-

It allow us to modify the existing data in database.

**Code:-**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

class Update{
    psvm (String args[]){

  try{
    Connection conn = DriverManager.getConnection("jdbc:mysql://
                      localHost:3306/Student", "root", "1234");

// Prepared statement and setting parameters

String update_data = "Update info SET name=? where id=?";
PreparedStatement pstmt = conn.prepareStatement(update_data);
pstmt.setString(1, "ADI");
pstmt.setInt(2,1);

int rowsUpdated = pstmt.executeUpdate();
if (rowsUpdated >0){
    System.out.println("Student's name was updated successfully!");
}}

  Catch (Exception e){
      System.out.println(e);}
}}
```

## (4) Delete (Delete operation):-

The delete operation allows us to remove data from the database.

Code:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

Class Delete{
   psvm (String args[]){

   try {

   Connection conn = DriverManager.getConnection("jdbc:mysql://local:3306/
                                                  student", "root", "1234");

   String delete_data = "Delete From info Where Id=?";

   //PreparedStatement and setting the parameter

   PreparedStatement pstmt = conn.prepareStatement(delete_data);

   // Execute the query
   int rowsDeleted = pstmt.executeUpdate();
    if (rowsDeleted>0){
       System.out.println("Student was deleted successfully!");
   }
   Catch (Exception e){
       System.out.println(e);}
}
```

## ★ Steps to use Prepared Statement

1) Create a SQL query with placeholders (?)
2) Set the parameter using:-
   a) For String → setString()
   B) For Integer → setInt()

3) Execute the query using executeUpdate() → Insert, Update, Delete
   Execute the query using executeQuery() → Select

## ★ CallableStatement Interface:-

The CallableStatement interface in JAVA is used to call stored procedure and functions from a database. These stored procedures or functions are SQL code that we store in the database and execute when we want.

Means, writing functions in MySQL same as we do in Java function.

### function create code (In MySQL side):-

```
mysql> Use Student; // Here student = database name

mysql> DELIMITER //
mysql> CREATE FUNCTION GetStudentName (StudentId INT)
    -> RETURNS VARCHAR(100)
    -> DETERMINISTIC
    -> READS SQL DATA
    -> BEGIN
    -> DECLARE StudentName Varchar(100);
```

→ Select name INTO StudentName From info where
id = studentId;
→ Return StudentName;
→ END //

mysql> DELIMITER ;

Java code:- (Calling function from mysql)

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;

Class RetrieveStudent {
    psvm (String args[]) {

    try {
        Connection conn = DriverManager.getConnection("jdbc:mysql://
                          localHost:3306/student", "root", "1234");

    // Create a callable statement for the function

    String call_data = "{ ? = CALL GetStudentName(?)}";
    CallableStatement call = conn.prepareCall (call_data);

    // Register the output parameter (name of the student)

    Call.registerOutParameter (1, Types.VARCHAR);

    // Set the input parameter (Student ID)
    Call.setInt (2, 1);
```

// Execute the function

```
Call.execute();

// Retrieve the output
String studentName = call.getString(1);
System.out.println("Student's name :" + studentName);}

Catch (Exception e){
    System.out.println(e);}

}}
```

★ **ResultSetMetaData Interface:-**

→ The ResultSetMetaData interface in Java is part of the JDBC API and is used to obtain information about the columns in a ResultSet.

→ This includes details such as the number of columns, the column names, data types, and the other properties.

○ When we run a Select query in database using JDBC, it returns a ResultSet, which contains the actual data. The ResultSetMetaData provides metadata about the columns of the ResultSet.

**Using of ResultSetMetaData:-**

→ Don't know the structure of result set in advance
→ Want to make program more dynamic, as result set could change in terms of column count or column names.

★ Methods in ResultSetMetaData:-

1) getColumnCount() → Returns the total number of columns.

2) getColumnName(int column) → Returns the name of the specified column.

3) getColumnTypeName(int column) → Returns the data type of the specified column.

4) getColumnDisplaySize(int column) → Returns the maximum number of characters for displaying the column.

5) isNullable(int column) → Check whether the column allows NULL values or not.

Code:- (ResultSetMetaData):-

```
import java.sql.*;

Class ResultSetMD {
  psvm(String args[]){

  try{
      Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/student", "root", "1234");

      String sql = "Select id, name From info";
      PreparedStatement pstmt = conn.prepareStatement(sql);
      ResultSet rs = pstmt.executeQuery();
```

```
// Get metadata of the ResultSet

ResultSetMetaData rsmd = rs.getMetaData();

// Get the numBer of columns

int columnCount = rsmd.getColumnCount();
System.out.println("Total columns :"+columnCount);


// Loop through each column to display metadata

for(int i=1; i<=columnCount; i++){

  System.out.println("Column "+i+":");
System.out.println("Name:"+rsmd.getColumnName(i));
System.out.println("Data Type:"+rsmd.getColumnTypeName(i));
System.out.println("Column Display size:"+rsmd.getColumn
                                          DisplaySize(i));

System.out.println("Is Nullable:"+rsmd.isNullable(i));
System.out.println();
}}

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