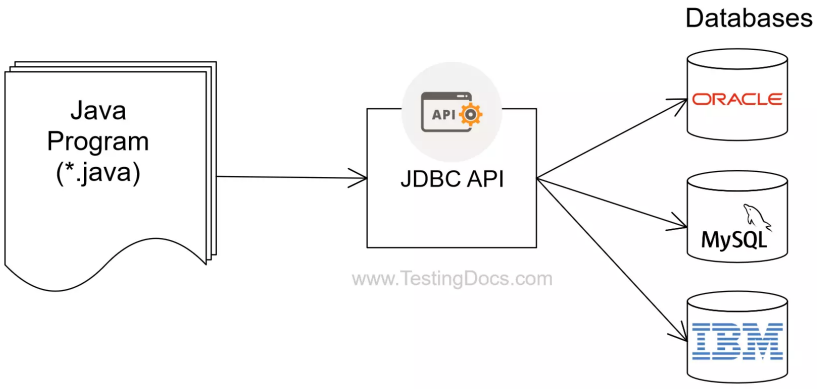
JAVA JDBC(Java Database Connectivity)

- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

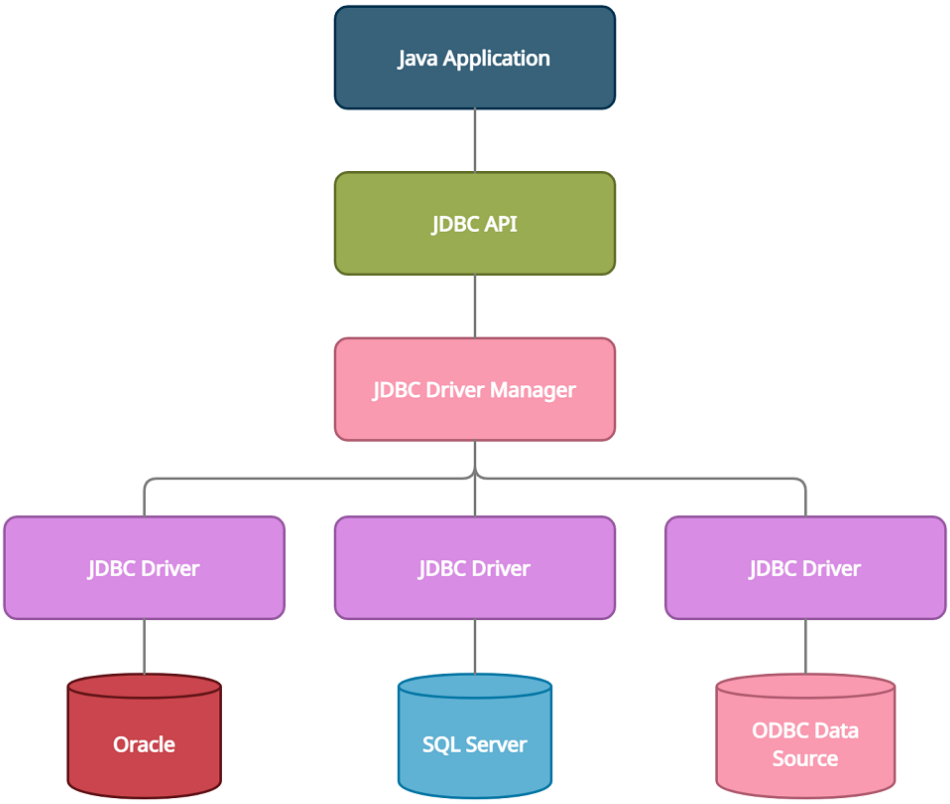
- JDBC API is used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

- JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.



- We have various types of Databases with us. Like MS Access, MySQL, Oracle, IBM etc..

- To communicate with with these databases - JAVA Programs/Applications need JDBC API.



The above figure shows the actual working of JDBC. Your java application needs JDBC API to connect with Database. For this We need JDBC Driver Manager that converts Java calls to DB specific calls and DB calls to Java specific calls.

1. **Application**: It is a java applet or a servlet that communicates with a data source.
2. **The JDBC API**: The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:
3. **DriverManager**: It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.

4. **JDBC drivers**: To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

- **What to import?** import java.sql.*;
- **What Exception need to be caught?** SQLException & ClassNotFoundException

Interface of JDBC API

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

Popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Types of JDBC Drivers

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

1) JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

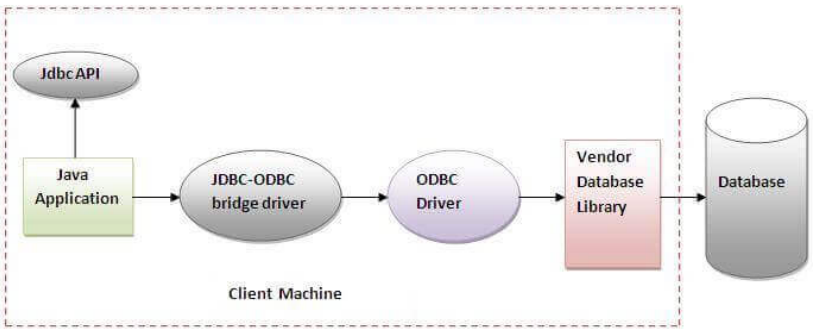


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.

- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

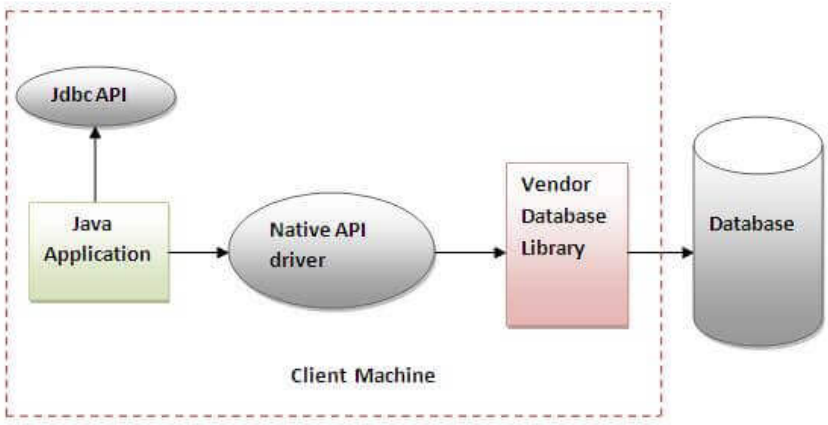


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

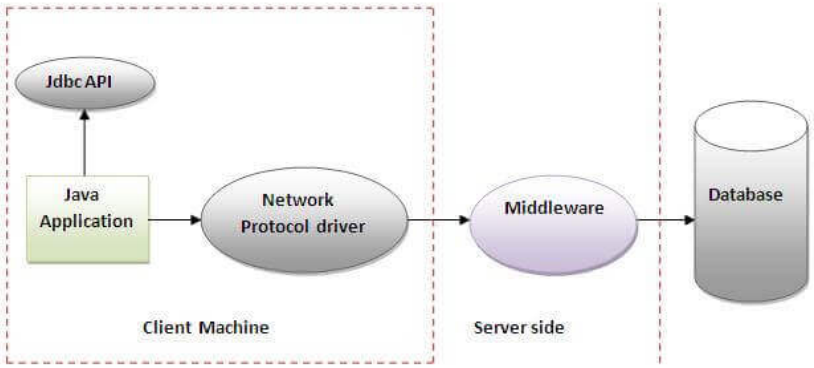


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language

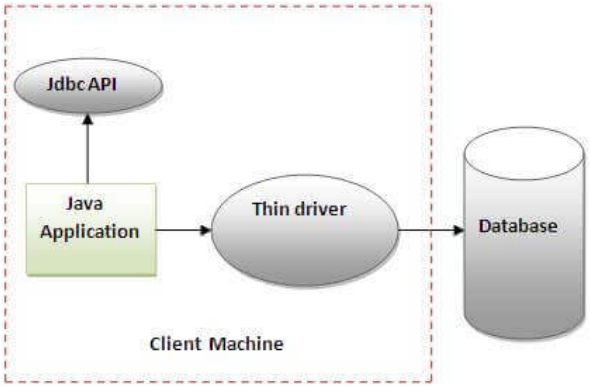


Figure- Thin Driver

Advantage:

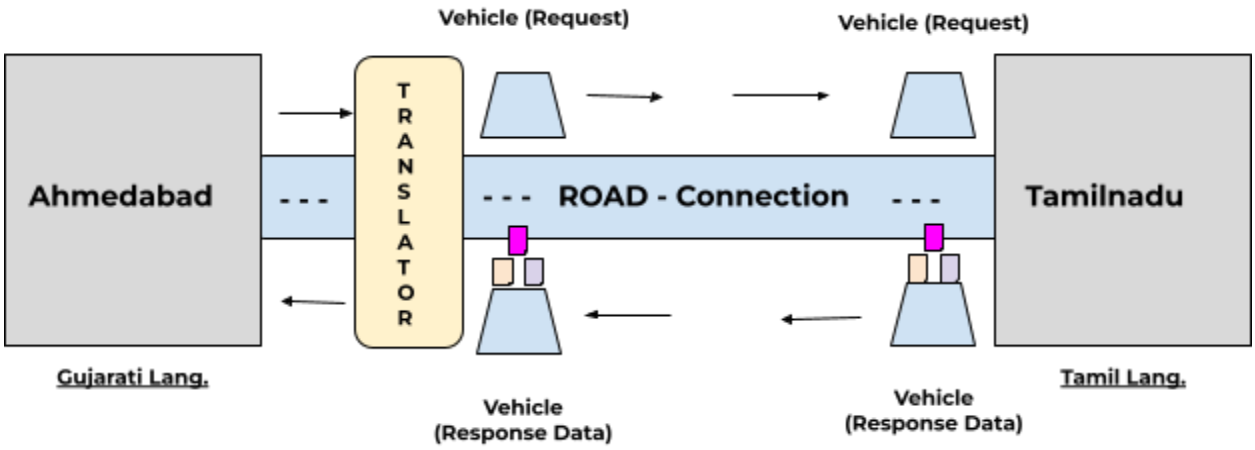
- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

UNDERSTAND THE JDBC CONNECTIVITY STEPS

- TO Understand the JDBC Connectivity concept and steps kindly consider the following concept.



Suppose in above scenario:

- One person from AHMEDABAD wants to buy goods from TAMILNADU. Now problem is that he does not know TAMIL language. So, First he need to identify the the translator. Which translates Gujarati to Tamil & vice versa.
- So, he decided following.

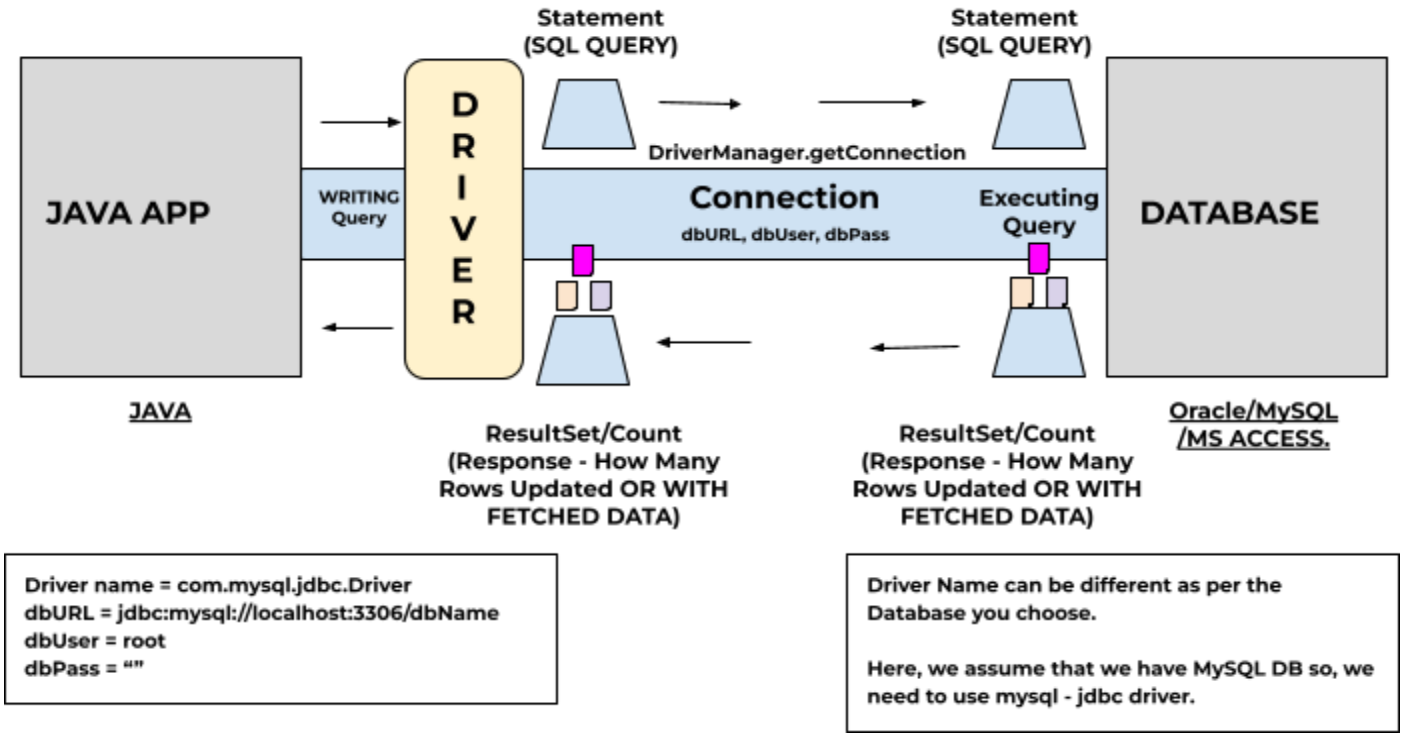
1. First Arrange the **Translator**
2. Develop the **Road** to Reach to the Tamilnadu. So he Developed the Road for **Connection**.
3. He get the **VEHICLE** and with it he shares the list of **Requirements.** & Via the established ROAD he send VEHICLE To TAMILNADU.

4. **Now,** At TAMILNADU the receiver understand the REQUIREMENTS & Loads the **GOODS in the VEHICLE.**
5. Now, Vehicle comes back with PROPER GOODS in the form of RESPONSE.
6. **First PERSON GET THE PROPER GOODS.**
7. **CLOSE THE ROAD.**

The above same Story can be applied to JDBC Example.

HOW?

See the below Figure & Try to map it with our above example.



To Execute Query we have 3 Methods.

1) **executeQuery(sql)** : executeQuery() command used for getting the data from database OR Select Query is used. It returns ResultSet Object. So we need to fetch data from it by iterating it.
2) **executeUpdate(sql)** command used for insert ,update, delete and Create & DROP query will be used here.
   It returns Integer value that shows how many rows get affected after applying insert/delete/update query.
3) **execute(sql):** used to apply any SQL Query. It returns boolean value.

PROBLEM

- At one end we have **JAVA APP** & on Other end WE HAVE - **DATABASE.**
- **JAVA APP** Runs on JAVA Language.
- DATABASE understands DB Specific languages like MySQL, ORACLE, IBM etc...
- So, we need to arrange the Translator first.
- Here **Translator** = **DRIVER**
- Here **Road** = **CONNECTION**
- Here **Vehicle** = **STATEMENT**
- Here **Request** = **SQL QUERY**
- Here **Response** = **FETCHED DATA/RESULT (RESULTSET)**

NOTE:

- So now, let the write the steps of JDBC COnnection with PROPER Syntax.
- Assume: Here we use MySQL DB - SO, need to install MySQL JDBC Driver.

STEPS FOR JDBC CONNECTIVITY

1. **Arrange & Load the Driver: Driver Name = com.mysql.jdbc.Driver**
   Syntax: Class.forName("com.mysql.jdbc.Driver");
2. **Do the connection with Database using Connection Interface & DriverManager class.**
   Syntax:

   String dbURL = "jdbc:mysql://localhost:3306/dbName";

   String dbUser = "root"; // default db username = root

   String dbPass = ""; // default password is null

   Connection con = DriverManager.getConnection(dbURL, dbUser, dbPass)

3. **Create Statement Object**

   Statement st = con.createStatement();

4. **Write SQL Query to apply on DB**

   String sql1 = "select * from tableName";

   String sql2 = "insert into tableName (col1, col2) values(val1, val2)";

5. **Execute Query using statement object. Now, if you do this by calling executeQuery() then it returns ResultSet Object & if you do by calling executeUpdate() then it returns the int value.**

```
int res = st.executeUpdate(sql1);

ResultSet rs = st.exectureQuery(sql);
```

6. **Extract the data from ResultSet**

```
while(rs.next())
{
        System.out. println("Data1 = "+rs.getInt(1));
        System.out. println("Data2 = "+rs.getString(2));
        System.out. println("Data3 = "+rs.getInt(3));
}
```

7. **Close the connection**

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

Note: Perform above steps to connect with Database and do the various operations,

PROGRAM 1 : Write a simple java code to establish a connection with Database using mysql jdbc Driver.

Assume: Database name = lj

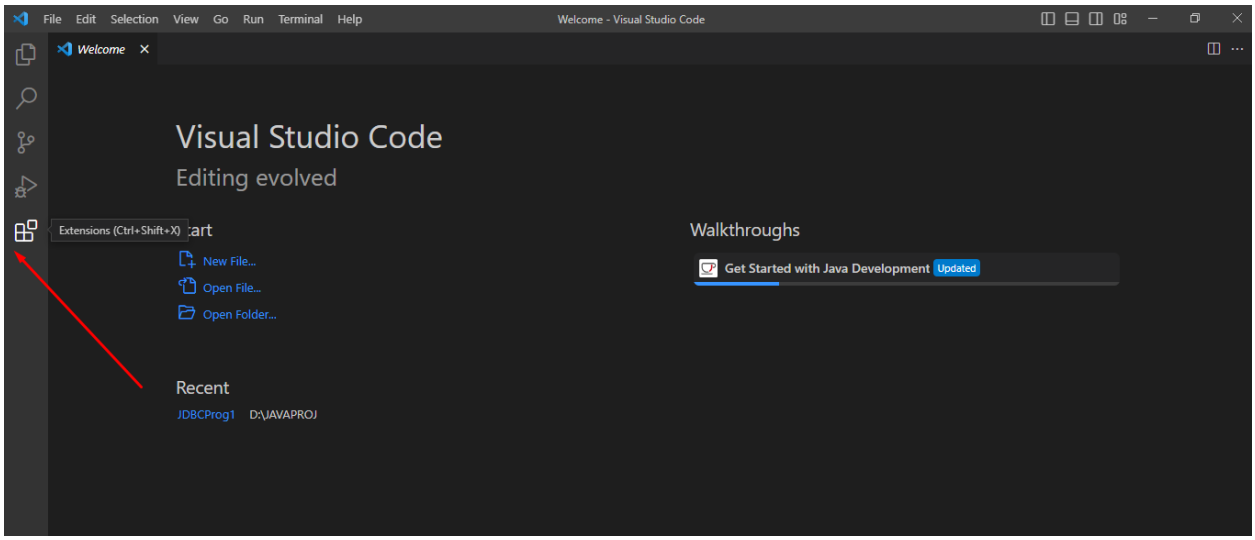
```
src > MyJDBC1.java > MyJDBC1 > main(String[])
1   import java.sql.*;
2   public class MyJDBC1 {
        Run | Debug
3       public static void main(String[] args) throws Exception{
4
5           //first we need to write the all variable necessary for connection
6           String dburl = "jdbc:mysql://localhost:3306/lj";
7           String dbuser = "root";
8           String dbpass = "";
9           String drivername = "com.mysql.jdbc.Driver";
10
11          // Step 1: Arrange and Load Drivers:
12
13          Class.forName(drivername);
14
15          // Step 2: Establish Connection:
16
17          Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);
18
19          // Check is Connection established?
20
21          if (con != null) {
22              System.out.println(x:"Connection Sucessful");
23          }
24
25          else {
26              System.out.println(x:"Connection Failed");
27          }
28      }
29  }
30
```

O/P

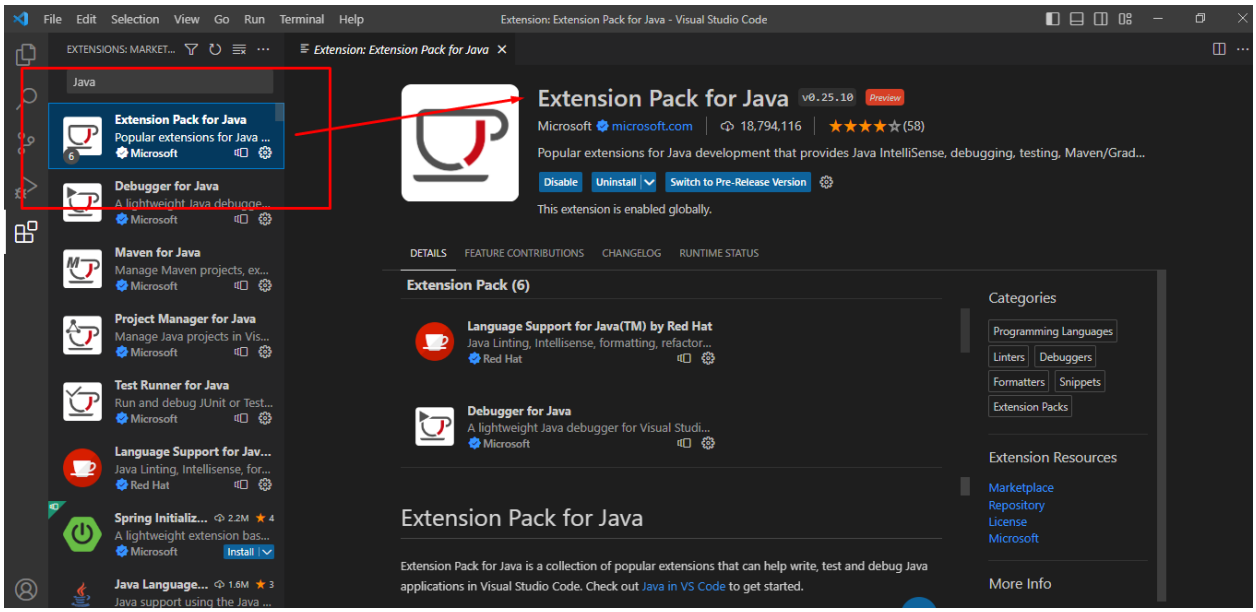
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Connection Sucessful
PS D:\jjs\JAVA JDBC\java project\JENIS>
```

HOW TO INSTALL JAVA IN VS CODE ?

1. Click on Extensions

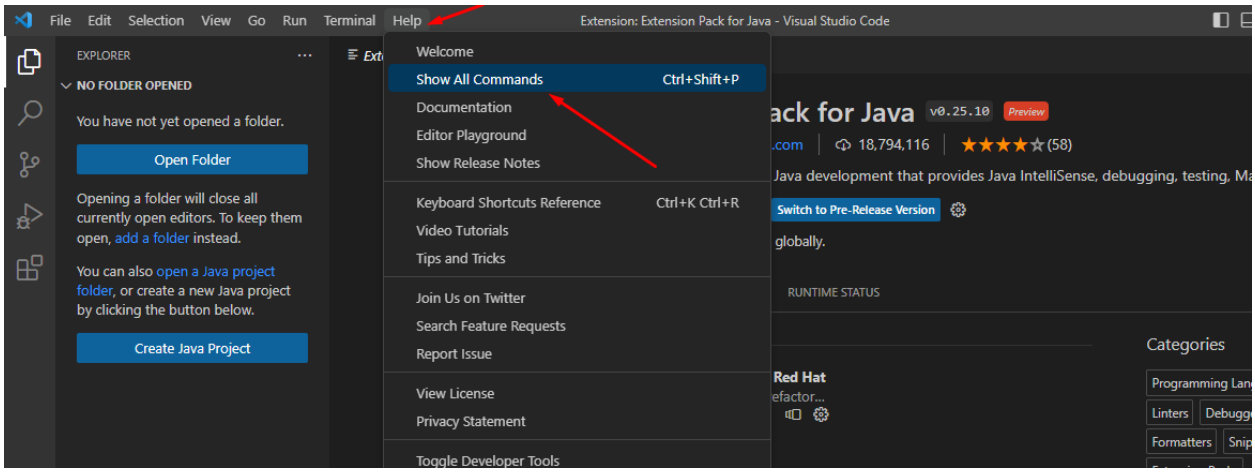


2. Search java in the search bar and select the following extension from the online. Make sure the internet should be live during this procedure.

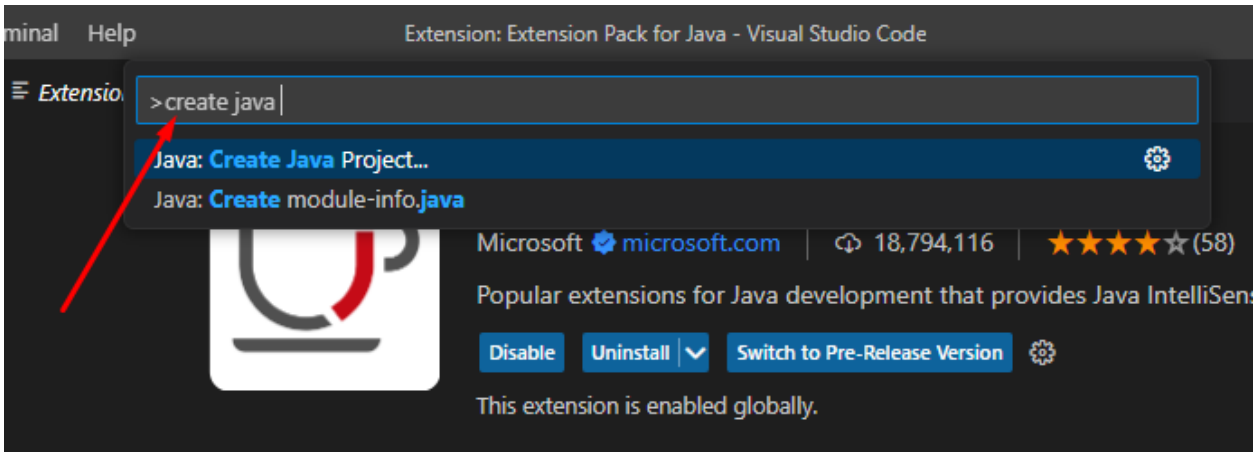


HOW TO CREATE JAVA PROJECT IN VS CODE FOR JDBC?

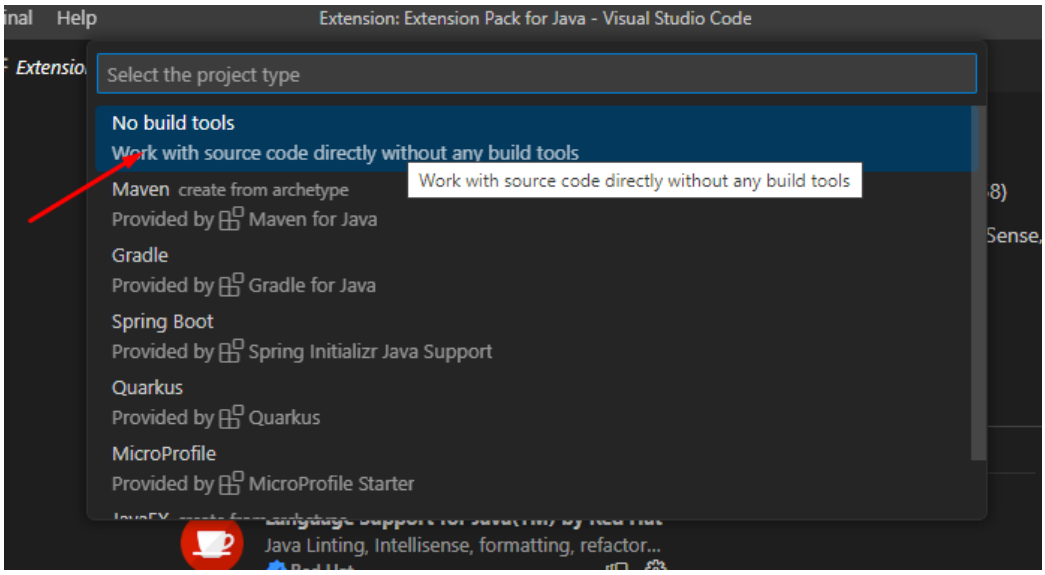
1. Click on Help
2. Click on show All Commands



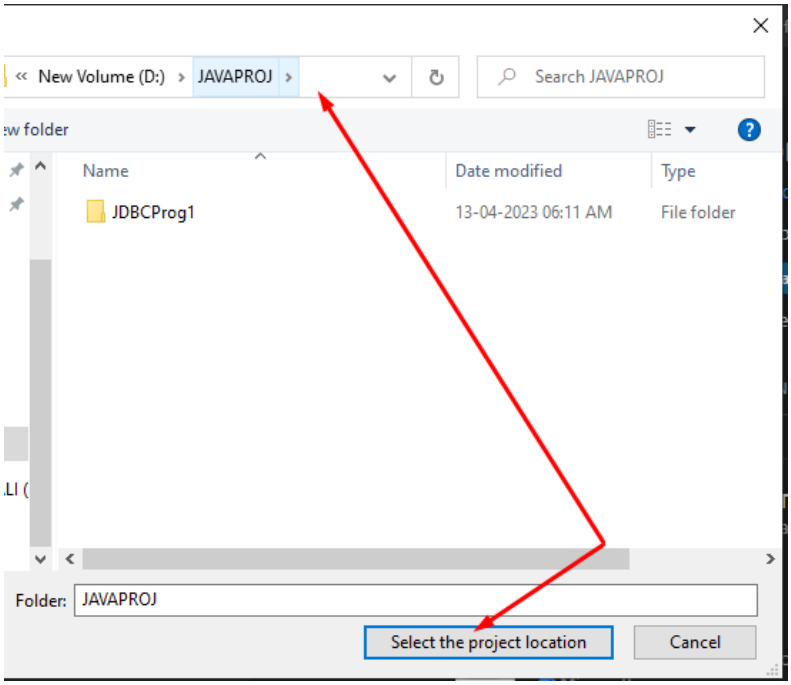
3. Write create java project in the search console



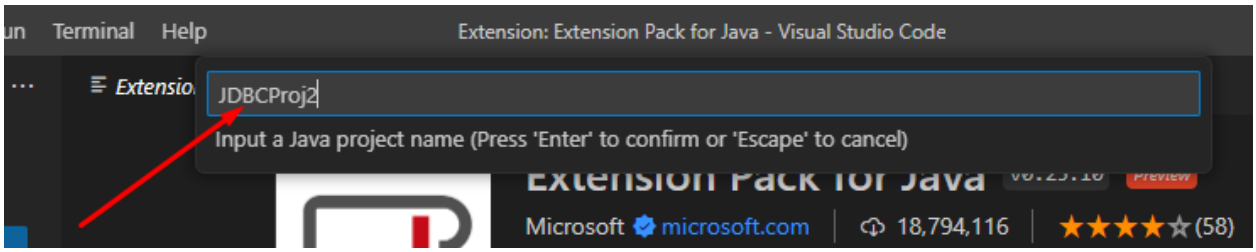
4. Choose No build tools



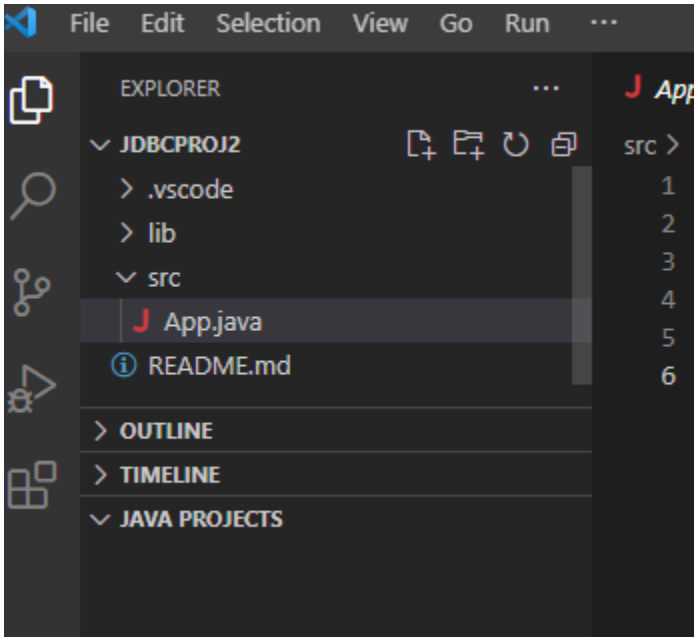
5. Select Project Location



6. Write Project Name over there : like here i wrote JDBCProj2 & Click Enter

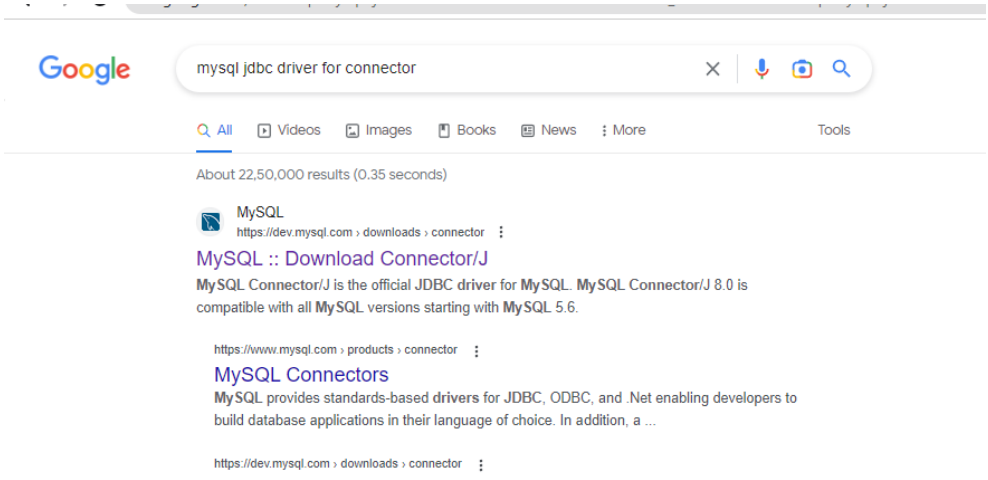


7. You can find folder structure like below.

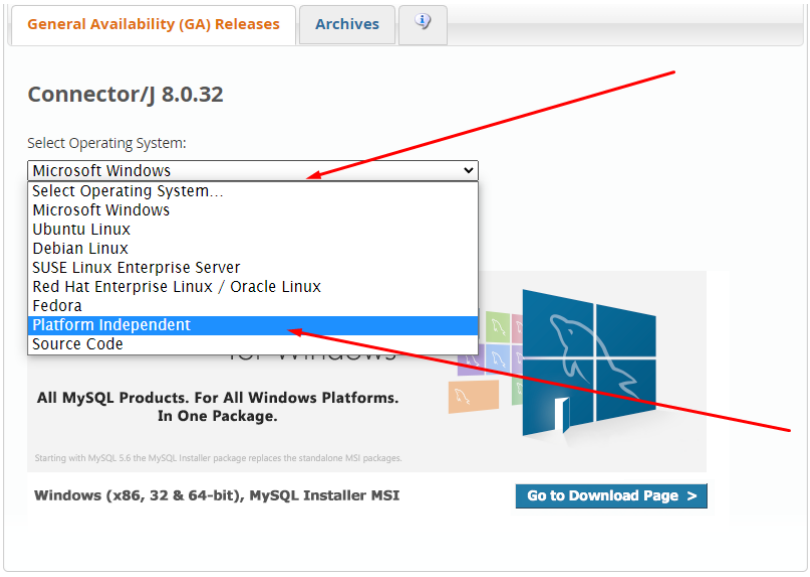


HOW TO ADD MySQL JDBC DRIVER IN YOUR JAVA PROJECT IN VS CODE FOR JDBC?

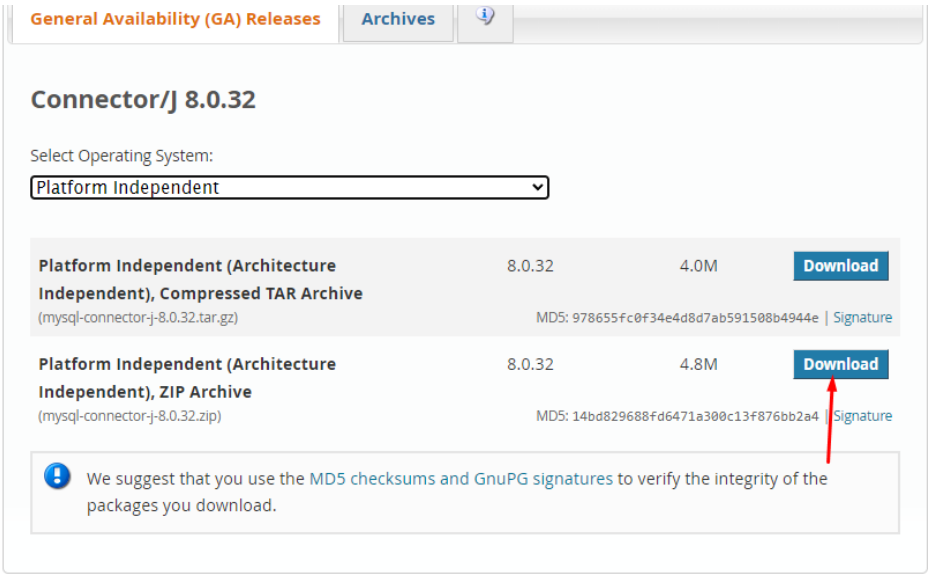
1. Write MySQL JDBC Driver in Google search



2. Click on : https://dev.mysql.com/downloads/connector/j/
3. Select Platform Independent



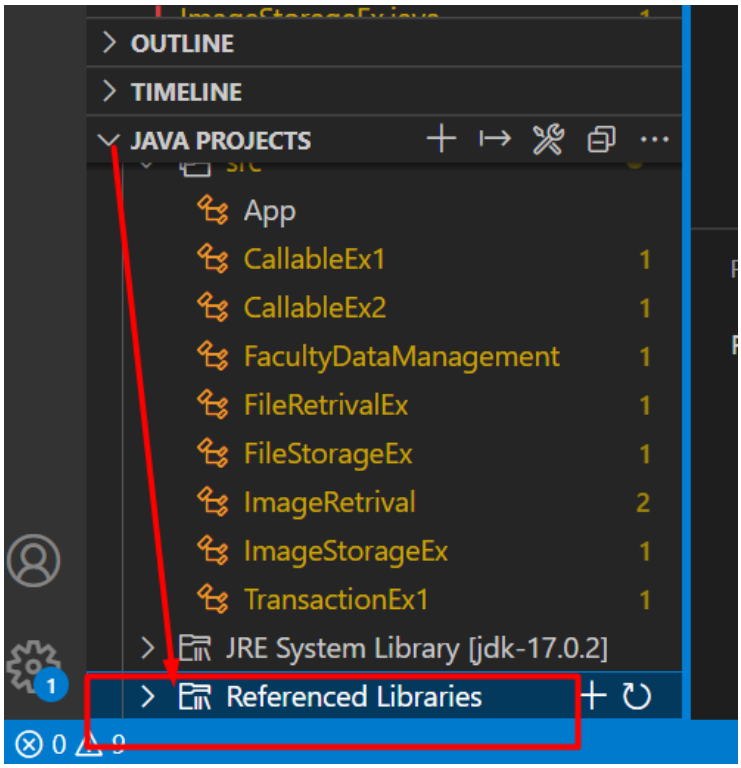
4. Download the zip file .



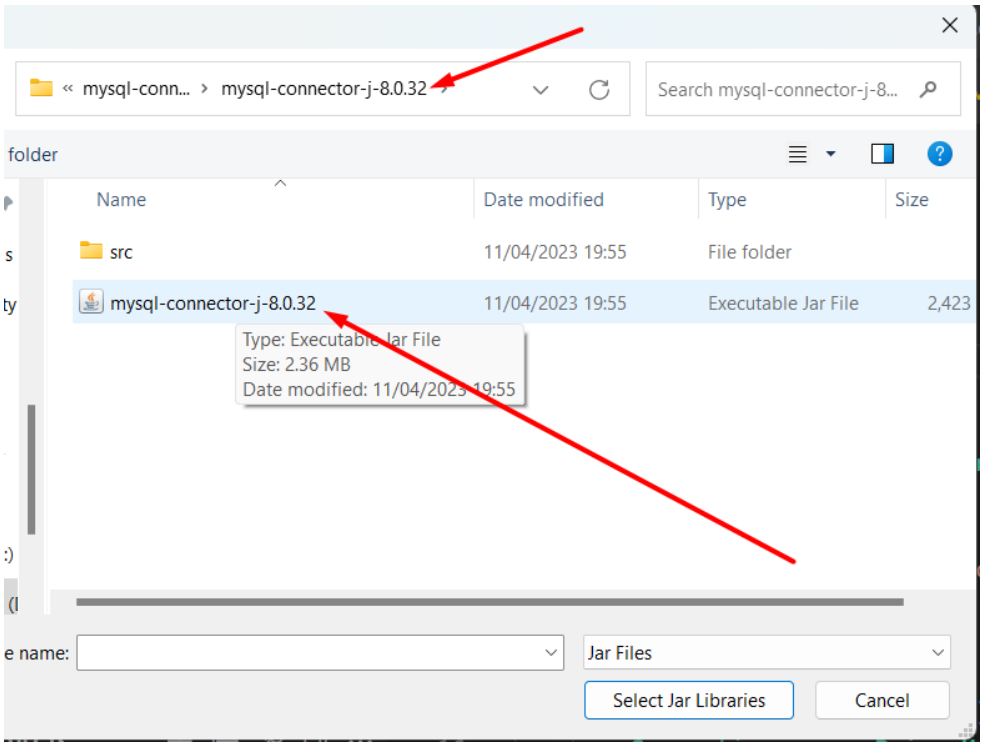
5. Import driver into your VS CODE JAVA PROJECT.

   Open Java Project ➔ **Click on Referenced Libraries**

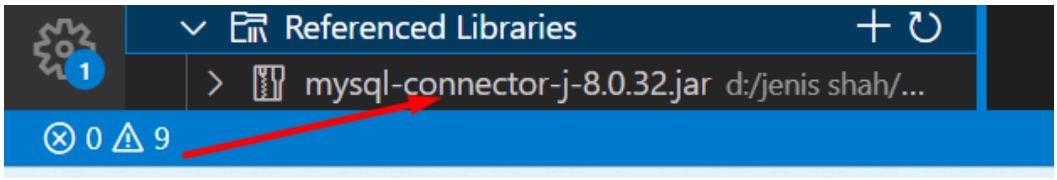
   Click on + Icon and Add Library



6. Import mysql.jdbc driver from your computer. Add .jar file only.



7. After importing drivers You can see the below added driver in your VS Code.



Difference Between executeUpdate(), executeQuery(), execute()

executeUpdate()	**executeQuery()**	**execute()**
executeUpdate() method used for update or modify database.	executeQuery() method used to retrieve some data from database.	execute() use for any SQL statements.
It returns an integer value. That shows the no. of rows affected in table	It returns an object of the ResultSet	It returns a boolean value. If it returns ResultSet object (working like executeQuery) then it returns TRUE . For int value(working like executeUpdate) it returns FALSE.
This method Is used to execute non SELECT queries. ● DML as INSERT, DELETE, UPDATE or ● DDL as CREATE. DROP	This method is normally used to execute SELECT queries	This method can be used to execute any type of SQL statement.
int i= st.executeUpdate(query);	ResultSet rs= st.executeQuery(query)	Boolean b= st.execute(query);

Mostly in this notes I have create a Database with Name = “lj” & table is “faculty”.

Database = lj

Table = faculty

Table Structure is as Below.

TABLE - faculty

Column Name	Column Type	Size	Is Primary Key?	Remarks
facId	INT	10	YES & Auto Incremented	
facName	VARCHAR	50		
salary	FLOAT	50		
number	VARCHAR	50		

INSERTION USING JDBC

- Establish the connection with database
- Write insert query

  Syntax: insert into tableName (col1, col2, col3) values (val1, val2, val3)
- Create Statement interface object.

  Statement st = con.createStatement(sql);
- Put the query in to Statement Interface and execute it.

  st.executeUpdate(sql)
- It returns the number rows inserted after executing queries.

PROGRAM 2:

```
import java.sql.*;

public class InsertJDBC {
   public static void main(String[] args) throws Exception {
      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:
      Class.forName(drivername);

      // Step 2: Establish Connection:
      Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);
      // Check is Connection established?
      if (con != null) {
         System.out.println("Connection Sucessful");
      }
      else {
         System.out.println("Connection Failed");
      }
      // Step 3: Create statement Object
      Statement st = con.createStatement();

      // Step 4: Write insert query for Faculty Table
      String sql = "insert into faculty(facId, facName, salary, number) values (1,
'JJS', 50000, '999898980')";
```

```
      // Step 5: Execute Query for insert, delete, update using executeUpdate()
and execute query select using executeQuery()
      int r = st.executeUpdate(sql); // This provides number of rows affected
      if (r > 0) {
         System.out.println("Insertion Success");
       }
       else {
         System.out.println("Insertion failed");
       }
   }
}
```

PROGRAM 3: Insert data in the table of faculty. But now this time do not use facId as it is Primary key so it is already AUTO INCREMENTED.
// Insert data without using primary Key (Due to auto increment)

```
import java.sql.*;

public class InsertJDBC2 {
   public static void main(String[] args) throws Exception {
      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:
      Class.forName(drivername);
```

```
        // Step 2: Establish Connection:
        Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

        // Check is Connection established?
        if (con != null) {
            System.out.println("Connection Sucessful");
        }
        else {
            System.out.println("Connection Failed");
        }

        // Step 3: Create statement Object
        Statement st = con.createStatement();

        // Step 4: Write insert query for Faculty Table
        String sql = "insert into faculty(facName, salary, number) values ('HDS',
55000, 9898989890)";

        // Step 5: Execute Query for insert, delete, update using executeUpdate()
and
        // execute query select using executeQuery()
        int r = st.executeUpdate(sql);
        if (r > 0) {
            System.out.println("Insertion Success");
        } else {
            System.out.println("Insertion failed");
        }
    }
}
```

PROGRAM 4: Insert data in the table of faculty. But now this time do not use facId as it is Primary key so it is already AUTO INCREMENTED. AS WELL AS ENTER THE VALUES FROM USER USING SCANNER CLASS.

// Insertion using User Input

```
import java.util.*;
import java.sql.*;

public class InsertJDBC3 {
   public static void main(String[] args) throws Exception {
      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      Scanner sc = new Scanner(System.in);
      System.out.println("How many rows you want to enter? ");
      int n = sc.nextInt();

      for (int i = 1; i <= n; i++) {
         System.out.println("Enter data for faculty " + i);
         System.out.println("Enter Faculty Name: ");
         String name = sc.next();
         System.out.println("Enter Salary: ");
         float sal = sc.nextFloat();
         System.out.println("Enter Number: ");

         String num = sc.next();
```

```
    // Step 1: Arrange and Load Drivers:
            Class.forName(drivername);
// Step 2: Establish Connection:
Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

            // Check is Connection established?

                if (con != null) {
                    System.out.println("Connection Sucessful");
                }
                else {
                    System.out.println("Connection Failed");
                }
                // Step 3: Create statement Object
                Statement st = con.createStatement();

// Step 4: Write insert query for Faculty Table
String sql = "insert into faculty (facName, salary, number) values ('" + name +
"'," + sal + ",'" + num + "')";
// Step 5: Execute Query for insert, delete, update using executeUpdate() and
// execute query select using executeQuery()

                st.executeUpdate(sql);
                 int r = st.executeUpdate(sql);
                 if (r > 0) {
                    System.out.println("Insertion Success of record "+i);
                 } else {
                    System.out.println("Insertion failed of recroed "+i);
                 }
```

```
            }
        }
}
```

DELETION USING JDBC

- Establish the connection with database
- Write Delete query
  Syntax: delete from tablename where col='val';
  delete from tableName where col1='val1' AND col2='val2'
- Create Statement interface object.
  Statement st = con.createStatement(sql);
- Put the query in to Statement Interface and execute it.
  st.executeUpdate(sql)
- It returns the number rows deleted after executing queries.

PROGRAM 5: Delete the data from the table of faculty.

//Deletion

```
import java.sql.*;

public class DeleteJDBC {
   public static void main(String[] args) throws Exception {
      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:
```

```
        Class.forName(drivername);
        // Step 2: Establish Connection:

       Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);
        // Check is Connection established?
        if (con != null) {
            System.out.println("Connection Sucessful");
        }
        else {
            System.out.println("Connection Failed");
        }
        // Step 3: Create statement Object
        Statement st = con.createStatement();
        // Step 4: Write insert query for Faculty Table
        String sql = "DELETE FROM faculty WHERE facName = 'DJU' or salary >=
55000";
        //String sql = "DELETE FROM faculty WHERE facName = 'sdjk'";
        //String sql = "DELETE FROM faculty WHERE facName = 'sdjk' AND
number = 'sfsd'";
 // Step 5: Execute Query for insert, delete, update using executeUpdate() and
// execute query select using executeQuery()
        int r;
        r = st.executeUpdate(sql);
        System.out.println("Rows affected: " + r);
        if (r > 0) {
            System.out.println("Deletion Success");
       } else {
            System.out.println("Deletion failed");
       }
```

```
    }
}
```

UPDATE USING JDBC

- Establish the connection with database
- Write update query

  Syntax:

  update tableName set col1='newVal' where col2='val';

  update tableName set col1='newVal1' , col2='newVal2' where col3='val';
- Create Statement interface object.

  Statement st = con.createStatement(sql);
- Put the query into Statement Interface and execute it.

  st.executeUpdate(sql)
- It returns the number rows updated after executing queries.

PROGRAM 5: UPDATE the data from the table of faculty.

```
import java.sql.*;
public class UpdateJDBC {
   public static void main(String[] args) throws Exception {

      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:
      Class.forName(drivername);
```

```
// Step 2: Establish Connection:
Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

// Check is Connection established?

if (con != null) {
   System.out.println("Connection Sucessful");
}
else {
   System.out.println("Connection Failed");
}

// Step 3: Create statement Object
Statement st = con.createStatement();
// Step 4: Write insert query for Faculty Table

//String sql = "update faculty set facName = 'DBP', salary = 35000";
//String sql = "update faculty set facName = 'DBP', salary = 35000 where facName = 'sdfs' or number = '2335'";
//String sql = "update faculty set facName = 'HDS' where facName = 'safs3' and salary = 2334";
//String sql = "update faculty set number = '9923467123', facName = 'JJS' where facName = 'sdfas'";
String sql = "UPDATE faculty SET facName = 'cvm'  WHERE facName = 'dbp'";
// String sql1 = "update faculty set facId = 4 where facId = 6";

// Step 5: Execute Query for insert, delete, update using executeUpdate() and
```

```
        // execute query select using executeQuery()

        int r;
        r = st.executeUpdate(sql);
        if (r > 0) {
            System.out.println("Updation Success");
            System.out.println("Rows affected: "+r);
        } else {
            System.out.println("Updation failed");
        }

    }
}
```

SELECT USING JDBC

- Establish the connection with database
- Write update query

  Syntax:

  select * from tableName;

  select col1, col2 from tableName;

  select col1, col2 from tableName where col3 = ‘val’;
- Create Statement interface object.

  Statement st = con.createStatement(sql);
- Put the query in to Statement Interface and execute it.

  st.executeQuery(sql)
- It returns the ResultSet Object. So retrieve data from it using while loop and various methods of ResultSet
- Ex:

  ResultSet rs = con.createStatement(sql);

```
while(rs.next())
{
        System.out.println("Data 1 = " + rs.getString(1)); // 1st column

        System.out.println("Data 2 = " + rs.getString("col2Name"));
        System.out.println("Data 3 = " + rs.getInt("col3name"));
        System.out.println("Data 4 = " + rs.getFloat("4")); // 4th column
}
```

PROGRAM 6: SELECT/FETCH the data from the table of faculty.

/*

Question : How to fetch Data from table?

--> Need to use: ResultSet

--> Execute query of: SELECT

--> Call Function: executeQuery (sql)

--> Iterate ResultSet using while loop

--> Apply following methods:

Here, one can pass col_index or col_name as argument

* **getInt(int col_index);**
* **getString (int col_index);**
* **getBoolean (int col_index);**

```
   * getFloat (int col_index);

   OR

   * getInt(String col_name);
   * getString (String col_name);
   * getBoolean (String col_name);
   * getFloat (String col_name);

*/
import java.sql.*;

public class SelectJDBC1 {
   public static void main(String[] args) throws Exception {

      String dburl = "jdbc:mysql://localhost:3306/Lou";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:

      Class.forName(drivername);

      // Step 2: Establish Connection:

      Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

      // Check is Connection established?
```

```
if (con != null) {
   System.out.println("Connection Sucessful");
}

else {
   System.out.println("Connection Failed");
}

// Step 3: Create statement Object

Statement st = con.createStatement();

// Step 4: Write insert query for Faculty Table

String sql = "SELECT * from faculty";
ResultSet rs = st.executeQuery(sql);
```

// Step 5: execute query select using executeQuery() --> Display Faculty Table

```
Used Column name as argument
while (rs.next()) {
   System.out.println("Faculty ID: " + rs.getInt("facId"));
   System.out.println("Faculty Name: " + rs.getString("facName"));
   System.out.println("Faculty Salary: " + rs.getFloat("salary"));
    System.out.println("Faculty Number: " + rs.getString("number"));
 }
```

```
        // Used Column index as argument --> Column index starts from 1.

        while (rs.next()) {
            System.out.println("Faculty ID: " + rs.getInt(1));
            System.out.println("Faculty Name: " + rs.getString(2));
            System.out.println("Faculty Salary: " + rs.getFloat(3));
            System.out.println("Faculty Number: " + rs.getString(4));
        }
    }
}
```

PROGRAM 7:// Select Specific Columns only from the table of faculty. using JDBC

```
import java.sql.*;

public class SelectJDBC2 {
    public static void main(String[] args) throws Exception {

        String dburl = "jdbc:mysql://localhost:3306/Lou";
        String dbuser = "root";
        String dbpass = "";
        String drivername = "com.mysql.jdbc.Driver";

        // Step 1: Arrange and Load Drivers:

        Class.forName(drivername);

        // Step 2: Establish Connection:
```

```
Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

// Check is Connection established?

if (con != null) {
   System.out.println("Connection Sucessful");
}

else {
   System.out.println("Connection Failed");
}

// Step 3: Create statement Object

Statement st = con.createStatement();

// Step 4: Write insert query for Faculty Table

//String sql = "SELECT facname, SALARY from faculty WHERE number =
'12345'";
String sql = "SELECT salary, facName from faculty WHERE number =
'12345'";

// After executing  above line
// Virtual table created, with salary at column index 1 and facName at
salary index 2
// So use this index for printing Data
```

```
        ResultSet rs = st.executeQuery(sql);

        // Step 5: execute query select using executeQuery() --> Display Faculty
Table

        // Used Column name as argument
        while (rs.next()) {

            System.out.println("Faculty Name: " + rs.getString(2));
            System.out.println("Faculty Salary: " + rs.getFloat(1));

            // System.out.println("Faculty Number: " + rs.getString("facName"));
            // System.out.println("Faculty ID: " + rs.getInt("salary"));
        }

        // Used Column index as argument


    }
}
```

PreparedStatement in JDBC

- A PreparedStatement is a pre-compiled SQL statement.
- It is a subinterface of Statement.
- Prepared Statement objects have some useful additional features than Statement objects.
- Instead of hard coding queries, PreparedStatement object provides a feature to execute a parameterized query.
- Ex: this is a parameterised Query :
- String sql="insert into emp values(?,?,?)";

Advantages of PreparedStatement

- When PreparedStatement is created, the SQL query is passed as a parameter.
- This Prepared Statement contains a pre-compiled SQL query, so when the PreparedStatement is executed, DBMS can just run the query instead of first compiling it.
- We can use the same PreparedStatement and supply with different parameters at the time of execution.
- An important advantage of PreparedStatements is that they prevent SQL injection attacks.

How to create instance of PrepatedStatement:

- public PreparedStatement prepareStatement(String query) throws SQLException{ }

Methods of PreparedStatement Interface

public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet

How to set data in the insert query?

String sql = “insert into faculty (facId, facName, salary, number) values(?,?,?,?)”;

NOTE:

- here, ? - is known as Placeholder. We will set value of it later by using set method.
- Query will be compiled once only.
- During execution query will run. It will not compile again. So this is faster execution than Statement.

```
PreparedStatement pst =  con.prepareCall(sql); //compiled once
pst.setInt(1,123);    // set value to 1st Placeholder
pst.setString(2,"JJS");
pst.setFloat(3,55000);
pst.setString(4,"12345678");
int  res = pst.executeUpdate(); // execute any number of  time
```

How to set data in the Update query?

Note: I want to update faculty salary = 50000 where faculty name = jjs

```
String sql = "update faculty set salary = ? where facName = ? ";
PreparedStatement pst =  con.prepareCall(sql);
pst.setFloat(1,50000); // here 1st place holder is for salary so we used Float and 1 as column index
pst.setString(2,"JJS"); // here 2nd place holder is for facName so we used String and 2 as column index
```

How to set data in the DELETE query?

Note: I want to delete faculty record where faculty name = jjs

```
String sql = "delete from faculty where  facName= ? ";
PreparedStatement pst =  con.prepareCall(sql);

pst.setString(1,"JJS"); // here Only one  place holder is for facName so we used String and 1 as column index
```

PROGRAM 8:// INSERT IN TABLE USING PREPARED STATEMENT

HERE NEW TABLE PROJECT - with columns pid, pname, ptype, pcost.

```
import java.sql.*;

public class PreparedStatEx1 {
   public static void main(String[] args) throws Exception {

      String dburl = "jdbc:mysql://localhost:3306/lj";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:

      Class.forName(drivername);

      // Step 2: Establish Connection:

      Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

      // Check is Connection established?

      if (con != null) {
         System.out.println("Connection Sucessful");
      }

      else {
         System.out.println("Connection Failed");
      }

      // Write Query for Insert

      String sql = "insert into project (ptype, pname, pcost) values (?,?,?)";

      // Create Prepared Statement

      PreparedStatement pst = con.prepareStatement(sql);
```

```
        pst.setString(1, "Research");
        pst.setString(2, "Manufacturing");
        pst.setFloat(3, 100000);

        System.out.println("Record Inserted: " + pst.executeUpdate());

        pst.setString(1, "Work");
        pst.setString(2, "Manu");
        pst.setFloat(3, 50000);

        System.out.println("Record Inserted: "+pst.execute());
    }
}
```

PROGRAM 9:// INSERT IN TABLE USING PREPARED STATEMENT WITH USER INPUT.

HERE NEW TABLE PROJECT - with columns pid, pname, ptype, pcost.

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

public class PreparedStatEx2 {
   public static void main(String[] args) throws Exception {

        String dburl = "jdbc:mysql://localhost:3306/lj";
        String dbuser = "root";
        String dbpass = "";
        String drivername = "com.mysql.jdbc.Driver";

        // Step 1: Arrange and Load Drivers:

        Class.forName(drivername);

        // Step 2: Establish Connection:

        Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);
```

```
        // Check is Connection established?

        if (con != null) {
            System.out.println("Connection Sucessful");
        }

        else {
            System.out.println("Connection Failed");
        }

        // Write Query for Insert

        String sql = "insert into project (ptype, pname, pcost) values (?,?,?)";

        // Create Prepared Statement

        PreparedStatement pst = con.prepareStatement(sql);

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Project Type: ");
        pst.setString(1, sc.next());

        System.out.println("Enter Project Name: ");
        pst.setString(2, sc.next());

        System.out.println("Enter Project Cost: ");
        pst.setFloat(3, sc.nextFloat());

        System.out.println("Record Inserted: " + pst.executeUpdate());

    }
}
```

PROGRAM 10:// UPDATE IN TABLE USING PREPARED STATEMENT HERE NEW TABLE PROJECT - with columns pid, pname, ptype, pcost.

```
import java.sql.*;

public class UpdatePrepared {
   public static void main(String[] args) throws Exception {

      String dburl = "jdbc:mysql://localhost:3306/Lou";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:

      Class.forName(drivername);

      // Step 2: Establish Connection:

      Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

      // Check is Connection established?

      if (con != null) {
         System.out.println("Connection Sucessful");
      }

      else {
         System.out.println("Connection Failed");
      }

      // Write Query for Insert

      String sql = "UPDATE project SET pname = ? WHERE pid = ?";

      // Create Prepared Statement

      PreparedStatement pst = con.prepareStatement(sql);
```

```
        pst.setString(1, "Manufacturing");
        pst.setInt(2, 518);

        pst.executeUpdate();
    }
}
```

PROGRAM 11:// DELETE IN TABLE USING PREPARED STATEMENT HERE NEW TABLE PROJECT - with columns pid, pname, ptype, pcost.

```
import java.sql.*;

public class DeletePrepared {
    public static void main(String[] args) throws Exception {

        String dburl = "jdbc:mysql://localhost:3306/Lou";
        String dbuser = "root";
        String dbpass = "";
        String drivername = "com.mysql.jdbc.Driver";

        // Step 1: Arrange and Load Drivers:

        Class.forName(drivername);

        // Step 2: Establish Connection:

        Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

        // Check is Connection established?

        if (con != null) {
            System.out.println("Connection Sucessful");
        }

        else {
            System.out.println("Connection Failed");
        }
```

```
        // Write Query for Delete

        String sql = "DELETE FROM project WHERE pid = ?";

        // Create Prepared Statement

        PreparedStatement pst = con.prepareStatement(sql);

        pst.setInt(2, 518);

        pst.executeUpdate();

    }
}
```

PROGRAM 12:// SELECT IN TABLE USING PREPARED STATEMENT HERE

NEW TABLE PROJECT - with columns pid, pname, ptype, pcost

```
import java.sql.*;

public class SelectPrepared {
    public static void main(String[] args) throws Exception {

        String dburl = "jdbc:mysql://localhost:3306/Lou";
        String dbuser = "root";
        String dbpass = "";
        String drivername = "com.mysql.jdbc.Driver";

        // Step 1: Arrange and Load Drivers:

        Class.forName(drivername);

        // Step 2: Establish Connection:

        Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

        // Check is Connection established?

        if (con != null) {
```

```
            System.out.println("Connection Sucessful");
        }

        else {
            System.out.println("Connection Failed");
        }

        // Write Query for Select

        String sql = "SELECT * FROM project WHERE ptype = ? AND pcost >= ?";

        // Create Prepared Statement

        PreparedStatement pst = con.prepareStatement(sql);

        pst.setString(1, "Work");
        pst.setInt(2, 50000);

        ResultSet rs = pst.executeQuery();

        while (rs.next()) {
            System.out.println("Project Type is: " + rs.getString("ptype"));
            System.out.println("Project Name is: " + rs.getString("pname"));
            System.out.println("Project Cost is: " + rs.getString("pcost"));
        }
        System.out.println(pst.execute());
    }
}
```

CallableStatement

- CallableStatement interface is used to call the stored procedures and functions.
- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- A Callable statement can have output parameters, input parameters, or both.
- The prepareCall() method of connection interface will be used to create CallableStatement object.

Difference Between Stored Procedures of DB and Functions of JAVA

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

How to Use callable Statement ?

1. Create Stored Procedure in DB using PHP MY ADMIN
2. Create CallableSatement object in java file to call that created stored procedure.

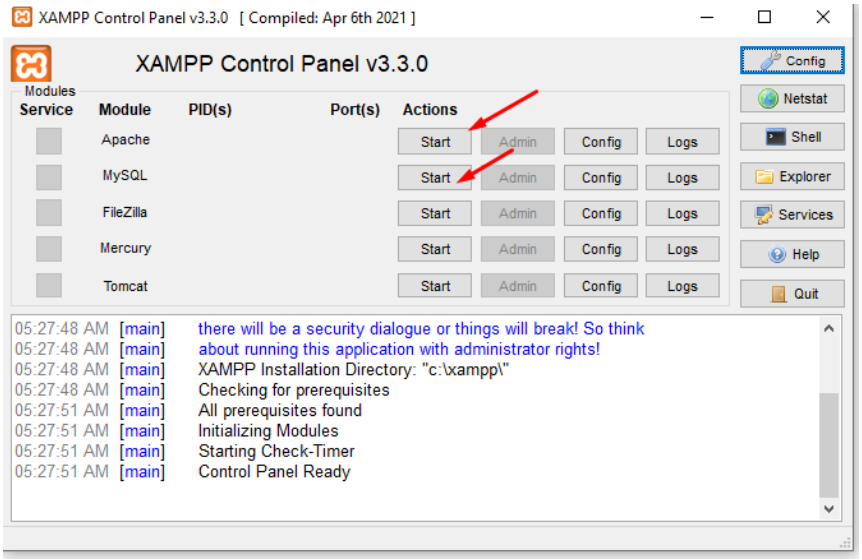
1. **How Create Stored Procedure in DB using PHP MY ADMIN?**

Syntax:

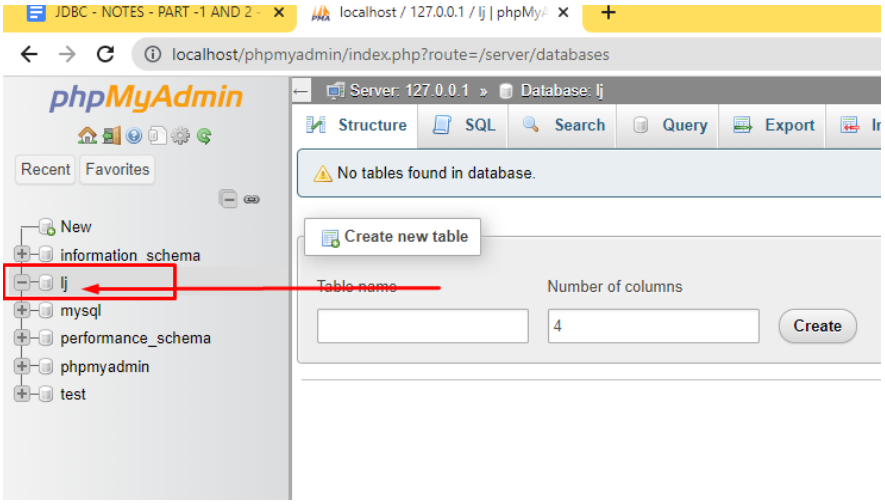
```
CREATE OR REPLACE PROCEDURE ProcedureName
( inVar2 IN column1, inVar2 IN column2,
 outVar1 INTO column3
)
AS

BEGIN
        SELECT column3
        FROM  tableName
        WHERE column1 = inVar1 AND column2 = inVar2;
END
```

STEPS TO FOLLOW IN PHP MYADMIN

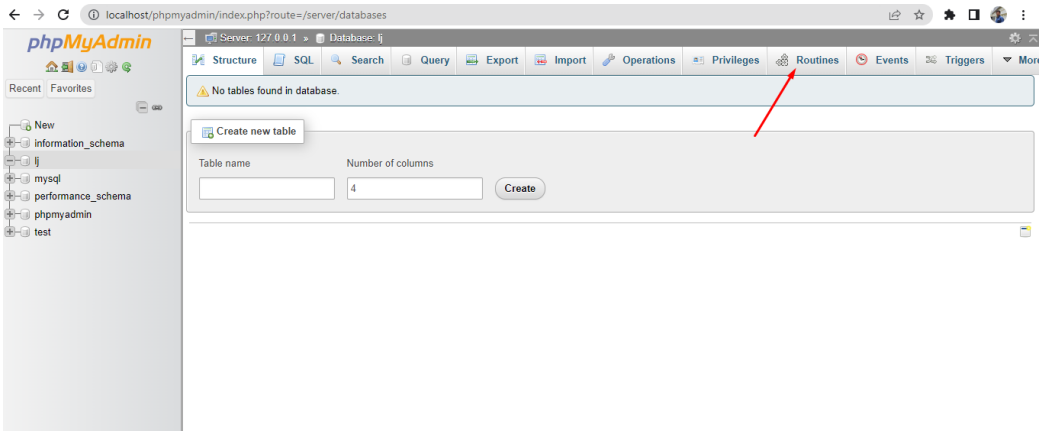
a) Open XAMPP & PhpMyAdmin



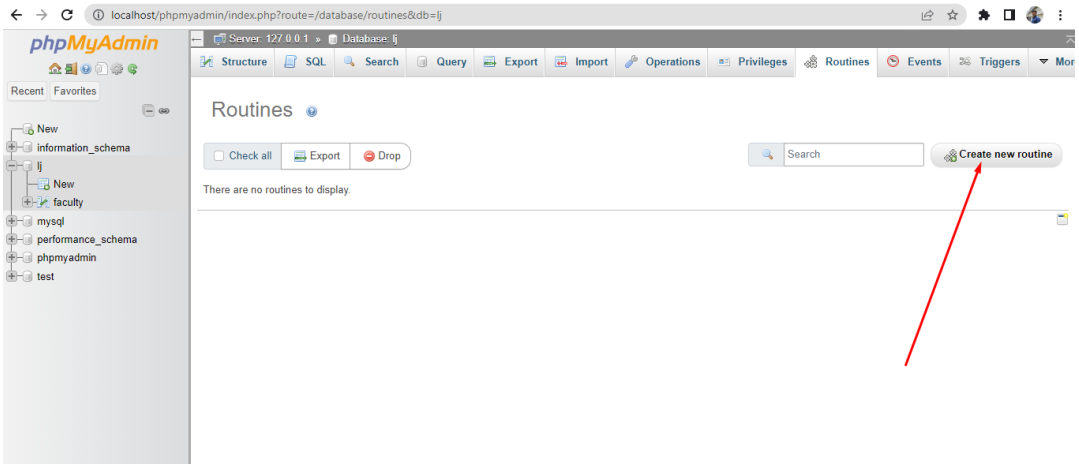
b) Click on your DB in which you want to create procedure



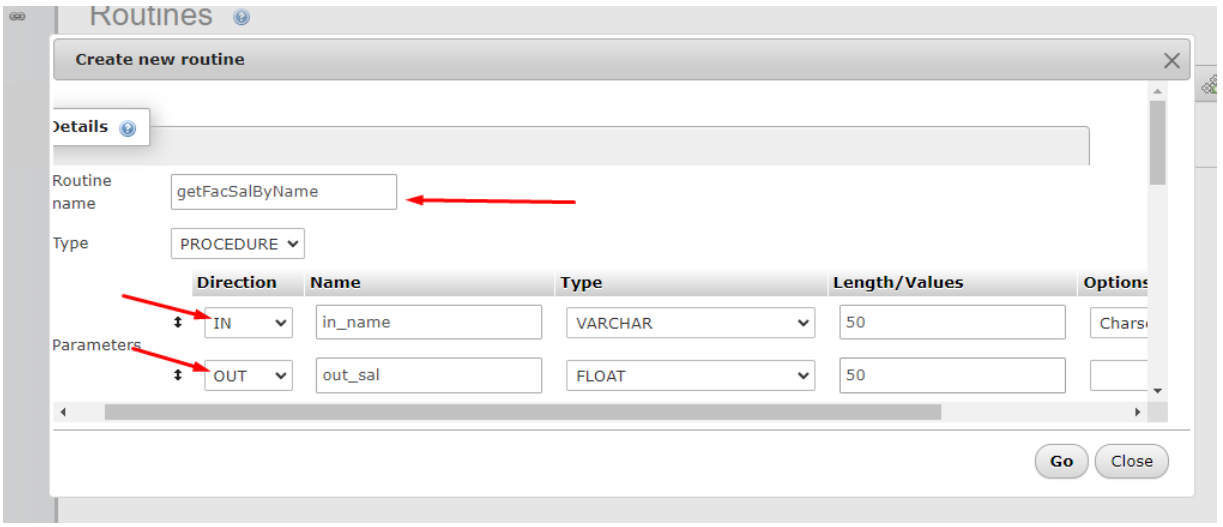
c) Click on Routine - Here Routine is the procedure.



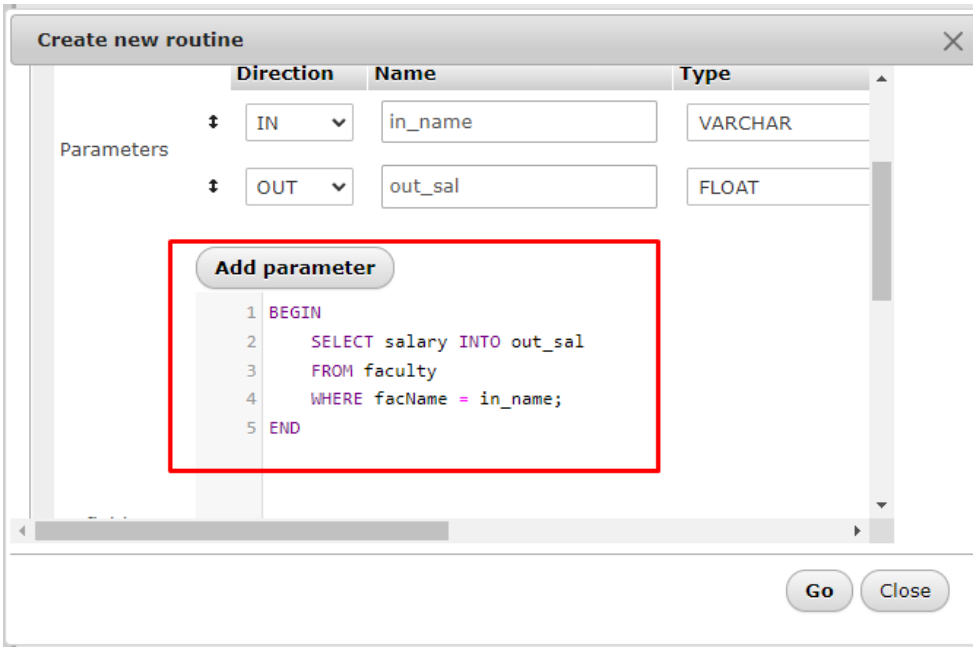
d) Add Routine & Write Routine Name / Procedure Name



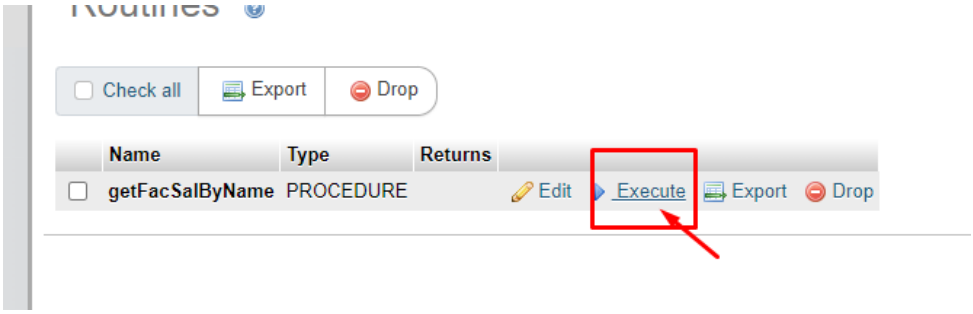
e) There is options for IN & OUT parameters. Here
IN - Means the values that you want to pass to PROCEDURE.
OUT - Means the values that you want to get from the PROCEDURE.
REMEMBER - WE CAN GET/RETURN MORE THAN ONE VALUE FROM THE PROCEDURE.

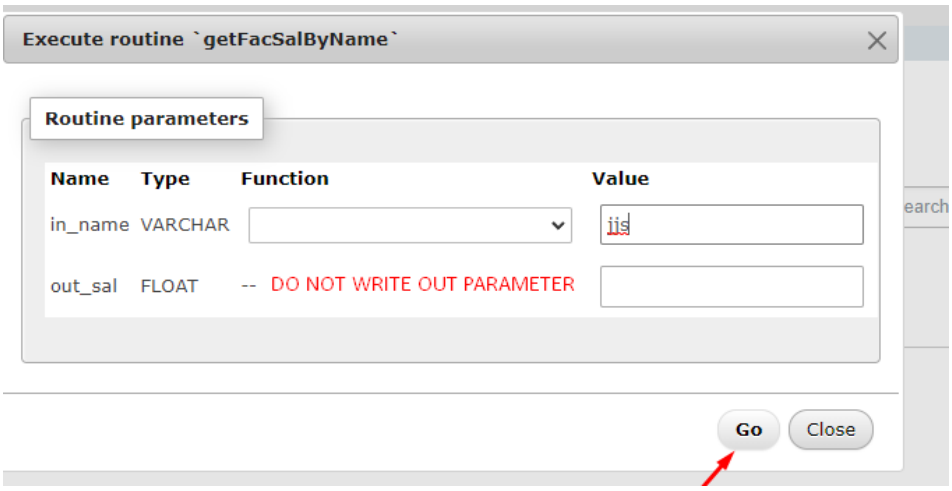


f) Write Procedure for it.
Ex: here i want to extract faculty salary from the procedure with name - **getFacSalByName - Here I pass name as In parameter.**

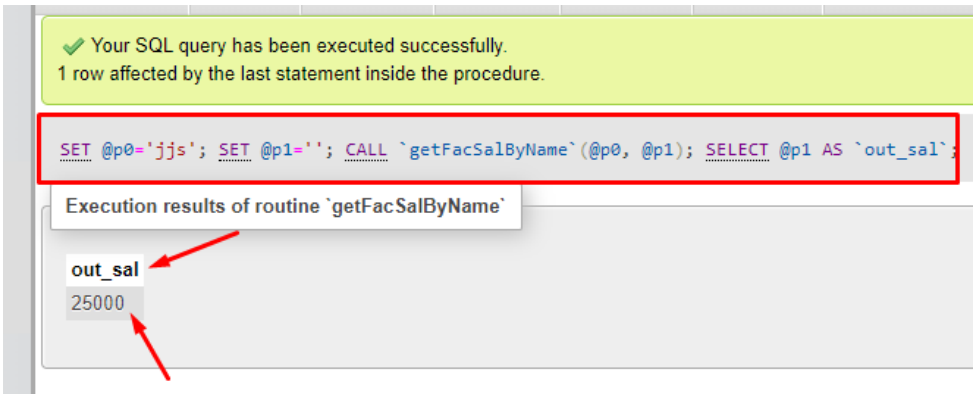


g) Click on Go. You can also check this by clicking on execute about procedure.





Output



2. Create CallableSatement object in java file to call that created stored procedure.

Java File.

```
Connection con = DriverManager.getConnection(dbURL, dbUser,
dbpass);
String sql = "{ call getFacSalByName (?,?) }";
CallableStatement cst = con.prepareCall(sql);

cst.setString(1,"jjs");  // first placeholder shows IN parameter
float  sal = cst.getFloat("salary");  // 2nd Placeholder shows OUT
Parameter

System.out. println("Salary = "+sal);
```

Program 13: // Simple callable example to fetch Faculty name from faculty table.

The procedure is as below:

```
CREATE PROCEDURE `getFacultyName`()
BEGIN
        SELECT facName
   FROM faculty;
END;
```

```
import java.sql.*;
import java.util.*;

public class CallableEx1 {
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";
```

```
        Class.forName(driverName);
        Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
        if(con!=null){
            System.out.println("DB Connection Success");
        }
        else{
            System.out.println("Connection Failed...");
        }

        String sql = " { call getFacultyName() }";
        CallableStatement cst = con.prepareCall(sql);
        ResultSet rs = cst.executeQuery(sql);
        while(rs.next())
        {
            System.out.println("Fac Name = "+ rs.getString(1));
        }

    }
}
```

Program 14: // callable example with IN and OUT parameter.

```
import java.sql.*;
import java.util.*;

// IN - means the parameter that need to pass from the procedure during
call
// OUT - the parameter that need to be fetched from the procedure after
call
```

The procedure is as below:

```
CREATE  PROCEDURE `getFacultyData`(IN `FAC_ID` INT(10), OUT
`FAC_NAME` VARCHAR(50))
BEGIN
        SELECT facName INTO FAC_NAME
   FROM faculty
   WHERE facId = FAC_ID;
END
```

```
public class CallableEx2 {
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }

      String sql = " { call getFacultyData(?,?) }  ";
      CallableStatement cst = con.prepareCall(sql);
      cst.setInt(  1, 3);
      cst.execute();
      String name = cst.getString(2);
      System.out.println("NAME = " + name);


   }
}
```

CRUD EXAMPLE

Program 15:

```
/*
Create CRUD Example with functions for every operations
Give choice to user:
1. Insert
2. Delete --> By Name
3. Update --> By Name
4. View --> 4.1 -> All Record
        4.2 -> Only App
        4.3 -> Only Website

 */

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

public class CRUD1 {

   static Connection con = null;

   static Scanner sc = new Scanner(System.in);

   static Statement st = null;

   // Insertion Method
   static void insertData() throws Exception {
      System.out.println("Enter Project Name: ");
      String pName = sc.next();
      System.out.println("Enter Type of Project: ");
      String pType = sc.next();
      System.out.println("Enter Cost of Project: ");
      float pcost = sc.nextFloat();

      String sql = "INSERT INTO Project (PNAME, ptype, pcost) values ('" +
pName + "','" + pType + "'," + pcost + ")";
```

```
        int r = st.executeUpdate(sql);

        if (r > 0) {
            System.out.println("Insertion Successful");
        } else {
            System.out.println("Insertion Failed");
        }
    }

    // Deletion Method
    static void deleteData() throws Exception {
        System.out.println("Enter Project Name you want to delete: ");
        String pName = sc.next();

        String sql = "DELETE FROM project WHERE pName ='" + pName + "'";

        int r = st.executeUpdate(sql);

        if (r > 0) {
            System.out.println("Deletion Successful");
        } else {
            System.out.println("Deletion Failed");
        }
    }

    // Updation Method
    static void updateData() throws Exception {
        System.out.println("Which Project's cost you want to update?");
        String pName = sc.next();
        System.out.println("Enter the New cost: ");
        float cost = sc.nextFloat();

        String sql = "UPDATE project SET pcost = '" + cost + "' WHERE pname = '"
+ pName + "'";
        int r = st.executeUpdate(sql);

        if (r > 0) {
            System.out.println("Updation Successful");
```

```
        } else {
            System.out.println("Updation failed");
        }
    }

    //View Method

    static void viewData() throws Exception {

        System.out.println("Enter 1 to view All Record\nEnter 2 to view App
Record\nEnter 3 to view Website Record");
        int temp = sc.nextInt();
        ResultSet rs;
        String sql;

        switch (temp) {
            case 1:
                sql = "SELECT * FROM project";
                rs = st.executeQuery(sql);

                while (rs.next()) {
                    System.out.println("Project Name: " + rs.getString("pname"));
                    System.out.println("Project Type: " + rs.getString("ptype"));
                    System.out.println("Project Cost: " + rs.getString("pcost"));
                }
                break;
            case 2:
                sql = "SELECT * FROM project WHERE ptype ='APP'";
                rs = st.executeQuery(sql);

                while (rs.next()) {
                    System.out.println("Project Name: " + rs.getString("pname"));
                    System.out.println("Project Type: " + rs.getString("ptype"));
                    System.out.println("Project Cost: " + rs.getString("pcost"));
                }
                break;
            case 3:
                sql = "SELECT * FROM project WHERE ptype ='WEBSITE'";
```

```
            rs = st.executeQuery(sql);

            while (rs.next()) {
               System.out.println("Project Name: " + rs.getString("pname"));
               System.out.println("Project Type: " + rs.getString("ptype"));
               System.out.println("Project Cost: " + rs.getString("pcost"));
            }
            break;
      }
   }

   public static void main(String[] args) throws Exception {

      String dburl = "jdbc:mysql://localhost:3306/Lou";
      String dbuser = "root";
      String dbpass = "";
      String drivername = "com.mysql.jdbc.Driver";
      Class.forName(drivername);

      con = DriverManager.getConnection(dburl, dbuser, dbpass);

      if (con != null) {
         System.out.println("Connection Successful");
      }
      else {
         System.out.println("Connection Failed");
      }

      st = con.createStatement();

      int ch;
      do {
         System.out.println("------------------------------ CRUD Operations
------------------------------");
         System.out.println(
            "Enter 1 for Insertion\nEnter 2 for Deletion\nEnter 3 for
Updation\nEnter 4 for View Data\nEnter 5 for Exit"
         );
```

```
            ch = sc.nextInt();

            switch(ch){
               case 1:
                  insertData();
                  break;
               case 2:
                  deleteData();
                  break;
               case 3:
                  updateData();
                  break;
               case 4:
                  viewData();
                  break;
            }
         } while (ch != 5);
      }
}
```

CHAPTER - 10

RESULTSET METADATA - INTERFACE

- The metadata means data about data i.e. we can get further information from the data.
- If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.

PROGRAM 16: Resultset metadata example

```
import java.util.*;
import java.sql.*;
public class RSMD {

   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }
      String sql = "select * from faculty";
      Statement st = con.createStatement();
      ResultSet rs = st.executeQuery(sql);

      ResultSetMetaData rsmd = rs.getMetaData();
      System.out.println("Total columns: "+rsmd.getColumnCount());
      System.out.println("Column Name of 1st column:
"+rsmd.getColumnName(1));
      System.out.println("Column Type Name of 1st column:
"+rsmd.getColumnTypeName(1));
      System.out.println("Table Name  "+rsmd.getTableName(1));

   }

}
```

O/P

```
DB Connection Success
Total columns: 4
Column Name of 1st column: facId
Column Type Name of 1st column: INT
Table Name  faculty
PS D:\jenis shah\My_E_Drive\LJ\LJU\JAVA-2 - Fac
```

DATABASE METADATA - INTERFACE

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

Commonly used methods of DatabaseMetaData interface

- **public String getDriverName()** throws SQLException: it returns the name of the JDBC driver.
- **public String getDriverVersion()**throws SQLException: it returns the version number of the JDBC driver.
- **public String getUserName()**throws SQLException: it returns the username of the database.
- **public String getDatabaseProductName()** throws SQLException: it returns the product name of the database.
- **public String getDatabaseProductVersion()** throws SQLException: it returns the product version of the database.
- **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)** throws SQLException: it

returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

Program 17: DatabaseMetaData Example

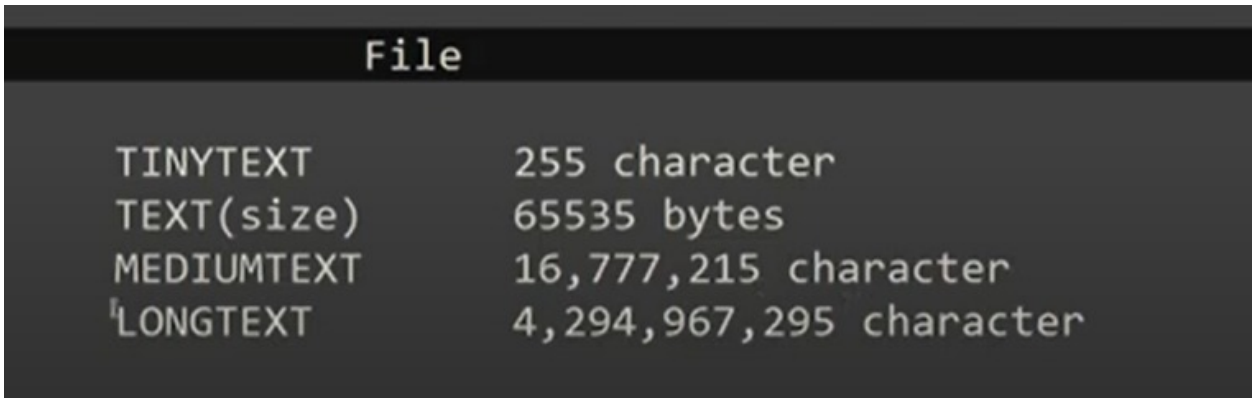
```
// Database MetaData

// Methods:
/*
   getDriverName()
   getDriverVersion()
   getUserName()
   getDatabaseProductName()
   getDtabaseProductVersion()
   getTables(catalog, patterns, tablepateern, String[] type)
 */

 import java.sql.*;

 public class DBMD {
    public static void main(String[] args) throws Exception {

        String dburl = "jdbc:mysql://localhost:3306/lju";
        String dbuser = "root";
        String dbpass = "";

        String driver = "com.mysql.jdbc.Driver";

        Class.forName(driver);

        Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

        if (con != null) {
            System.out.println("Connection Sucessful");
        } else {
            System.out.println("Connection Failed");
        }
```

```
        DatabaseMetaData dbmd = con.getMetaData();
        System.out.println("Driver Name: " + dbmd.getDriverName());
        System.out.println("Driver Version: " + dbmd.getDriverVersion());
        System.out.println("UserName is: " + dbmd.getUserName());
        System.out.println("Database Product Name: " +
dbmd.getDatabaseProductName());
        System.out.println("Database Product Version: " +
dbmd.getDatabaseProductVersion());

        String table[] = { "TABLE" };
        ResultSet rs = dbmd.getTables(null, null, null, table);
        while (rs.next()) {
           System.out.println("Table Name = "+rs.getString(3));
        }

    }
}
```

O/P

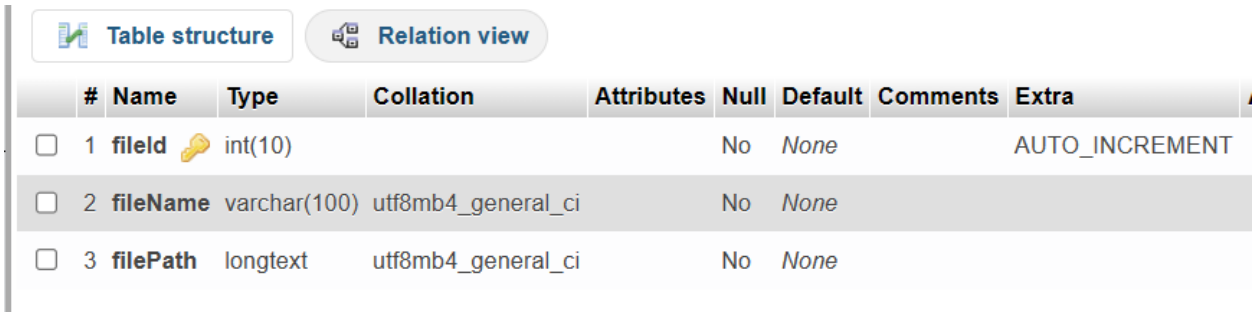
```
Connection Sucessful
Driver Name: MySQL Connector/J
Driver Version: mysql-connector-j-8.0.32 (Revision: fa4912a849140828e48162a2c396c8df0091bed7)
UserName is: root@localhost
Database Product Name: MySQL
Database Product Version: 5.5.5-10.4.22-MariaDB
Table Name = binaryfiletable
Table Name = faculty
Table Name = filetable
Table Name = pma__bookmark
Table Name = pma__central_columns
```

FILE STORAGE & RETRIEVAL IN JAVA JDBC



PROGRAM 18:
We need to store content of file into the database.

Table : filetable



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	fileId	int(10)			No	None		AUTO_INCREMENT
2	fileName	varchar(100)	utf8mb4_general_ci		No	None		
3	filePath	longtext	utf8mb4_general_ci		No	None		

```
import java.util.*;
import java.sql.*;
import java.io.*;


public class FileStorageEx{
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
```

```
        String dbUser = "root";
        String dbPass = "";

        Class.forName(driverName);
        Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
        if(con!=null){
           System.out.println("DB Connection Success");
        }
        else{
           System.out.println("Connection Failed.    ..");
        }

        String sql = "insert into filetable values(?,?,?)";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setInt(1, 1);
        pst.setString(2, "ATM JAVA FILE");
        File f = new File("D://ATM.java"); //
        FileReader fr = new FileReader(f);
        pst.setCharacterStream(3, fr,f.length());
        int res = pst.executeUpdate();
        if(res>0){
           System.out.println("File storage success");
        }
        else{
           System.out.println("File Storage Failed");
        }

    }
}
```

PROGRAM 19:

We need to fetch content of file from the database & create new file.

Table : filetable is same as above.

```
import java.util.*;
import java.sql.*;
```

```
import java.io.*;


public class FileRetrivalEx{
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }

      String sql = "select * from fileTable";
      PreparedStatement pst = con.prepareStatement(sql);
      ResultSet rs = pst.executeQuery();

      while(rs.next())
      {
         String fileName = rs.getString("fileName");
         Clob fClob = rs.getClob("filePath");
         Reader r = fClob.getCharacterStream();

         FileWriter fw = new FileWriter("D://"+fileName+".txt");
         int i;
         while((i = r.read())!=-1)
         {
            fw.write((char)i);
         }
         System.out.println("D://"+fileName+" saved");
```

```
            fw.close();
        }

    }
}
```

BINARY FILE STORAGE & RETRIEVAL IN JAVA JDBC (Binary file = Image, audio & video)

BLOB : Binary Long Object
A BLOB is binary large object that can hold a variable amount of data

Type of BLOB	Maximum amount of Data that can be stored
TINYBLOB	Up to 255 bytes
BLOB	Up to 64 Kb
MEDIUMBLOB	Up to 16 Mb
LONGBLOB	Up to 4 Gb

PROGRAM 20: Store image in the database.
Here tableName = binaryfiletable
The fileContent type is : LongBlob

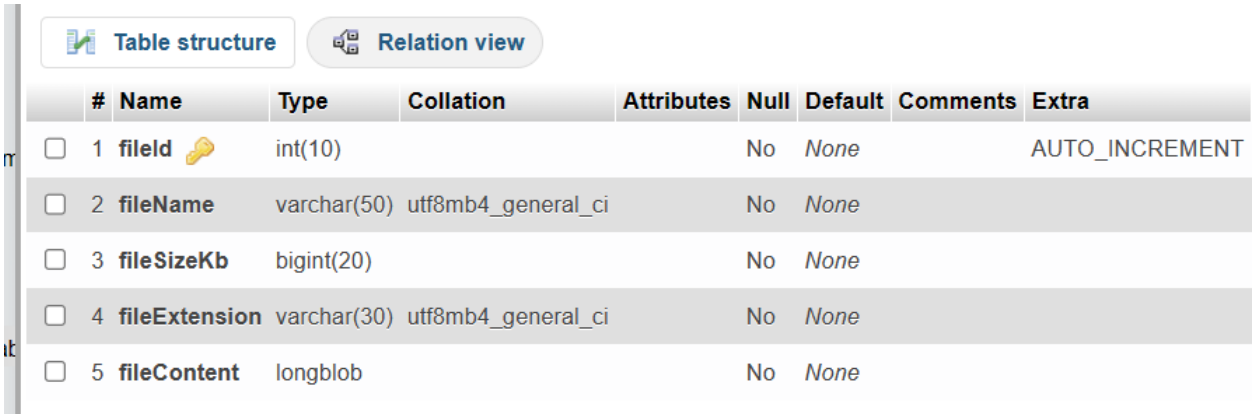


Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	fileId	int(10)			No	None		AUTO_INCREMENT
2	fileName	varchar(50)	utf8mb4_general_ci		No	None		
3	fileSizeKb	bigint(20)			No	None		
4	fileExtension	varchar(30)	utf8mb4_general_ci		No	None		
5	fileContent	longblob			No	None		

```
import java.util.*;
import java.sql.*;
import java.io.*;

public class ImageStorageEx {
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }


      String sql = "insert into binaryfiletable (fileName, fileSizeKb,
fileExtension, fileContent) values(?,?,?,?)";
      PreparedStatement pst = con.prepareStatement(sql);
      File f = new File("D://LOGO-01.png");
```

```
        //get all four values of parameters
        String fileName = f.getName();
        long fileLength = f.length();
        long fileSizeKb = fileLength/1024;
        String fileExtension = fileName.substring(fileName.lastIndexOf(".")+1);


         // Now set the values in query.
        pst.setString(1, fileName);
        pst.setLong(2, fileSizeKb);
        pst.setString(3, fileExtension);

        // here the all files need to be written in Binary So we need
FileInputStream
        FileInputStream fis = new FileInputStream(f);
        pst.setBinaryStream(4, fis,fileLength);
        int res = pst.executeUpdate();
        if(res>0){
           System.out.println("Binary File storage success");
        }
        else{
           System.out.println("Binary  File Storage Failed");
        }

    }
}
```

PROGRAM 21: Retrieve image in the database.
Here tableName = binaryfiletable
The fileContent type is : LongBlob

```
 /*
 * public Blob getBlob()throws SQLException
Signature of getBytes() method of Blob interface
public  byte[] getBytes(long pos, int length)throws SQLException
 *
 */
```

```
import java.util.*;

import javax.swing.plaf.FontUIResource;

import java.sql.*;
import java.io.*;

public class ImageRetrival {
   public static void main(String[] args) throws Exception {

      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }


      String sql = "select * from binaryfiletable";
      PreparedStatement pst = con.prepareStatement(sql);
      ResultSet rs = pst.executeQuery();

      while(rs.next())
      {
         Blob b = rs.getBlob("fileContent");
         byte arr[] = b.getBytes(1, (int)b.length());
         String fileName="D://new"+rs.getString("fileName");
         FileOutputStream fos = new FileOutputStream(fileName);
         fos.write(arr);
```

```
            fos.close();

        }
        con.close();

    }
}
```

TRANSACTION IN DATABASE

Transaction represents a single unit of work.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Transaction Example

```
import java.sql.*;
import java.util.*;

public class TransactionEx1 {
   public static void main(String[] args) throws Exception {
      String driverName = "com.mysql.jdbc.Driver";
      String dbURL = "jdbc:mysql://localhost:3306/lju";
      String dbUser = "root";
      String dbPass = "";

      Class.forName(driverName);
      Connection con = DriverManager.getConnection(dbURL, dbUser,
dbPass);
      if(con!=null){
         System.out.println("DB Connection Success");
      }
      else{
         System.out.println("Connection Failed.    ..");
      }
      con.setAutoCommit(false);
      String sql = "insert into faculty (facName, salary, number) values('KKKK',
123456, 456789)";
      PreparedStatement pst = con.prepareStatement(sql);
      int res = pst.executeUpdate();
      if(res>0){
         System.out.println("Insertion success");
      }
      else{
         System.out.println("Insertion Failed");
      }
      con.commit();
      con.close();

   }

}
```

Transaction Ex : 2 :

```
import java.sql.*;
import java.io.*;
import java.util.*;

public class TransactionManagement {
   public static void main(String[] args) throws Exception {
      String dburl = "jdbc:mysql://localhost:3306/Lju";
      String dbuser = "root";
      String dbpass = "";
      // String drivername = "com.mysql.cj.jdbc.Driver";

      // Step 1: Arrange and Load Drivers:

      // Class.forName(drivername);

      // Step 2: Establish Connection:

      Connection con = DriverManager.getConnection(dburl, dbuser, dbpass);

      // Check is Connection established?

      if (con != null) {
         System.out.println("Connection Sucessful");
      } else {
         System.out.println("Connection Failed");
      }

      con.setAutoCommit(false);

      String sql = "INSERT INTO faculty (fname, fsalary, fnumber) VALUES
('jenis',234500,'6748576896')";

      PreparedStatement pst = con.prepareStatement(sql);
```

```
        int r = pst.executeUpdate();

        String s = (r > 0) ? "Insertion Successful" : "Insertion Failed";

        System.out.println(s);

        Scanner sc = new Scanner(System.in);

        int temp;

        do {
            System.out.println("Enter 1 for Commit\nEnter 2 for Rollback");
            temp = sc.nextInt();
            switch (temp) {
                case 1:
                    con.commit();
                    System.out.println("commit");
                    break;
                case 2:
                    con.rollback();
                    System.out.println("rollback");
                    break;
                default:
                    System.out.println("ReEnter Option from 1, 2 or 3");
                    break;
            }
        } while (temp != 3);
    }
}
```