



# JDBC

## Java Database Connectivity

### Topics Covered:-

- JDBC
- Why we should we use JDBC
- JDBC Drivers
- JDBC API
- Java Database Connectivity with MySQL
- Steps to Connect with Database
- Creating Connection with database (JDBC)
- Creating Table (CRUD)
- Inserting values into table (CRUD)
- *Dynamic Values inserted*
- Updating data using the dynamic values using BufferedReader
- Fetching details from database (CRUD)
- Data Access Object in JDBC - (DAO classes)

 **Uday Sharma**

 **mrudaysharma4600@gmail.com**

# JDBC

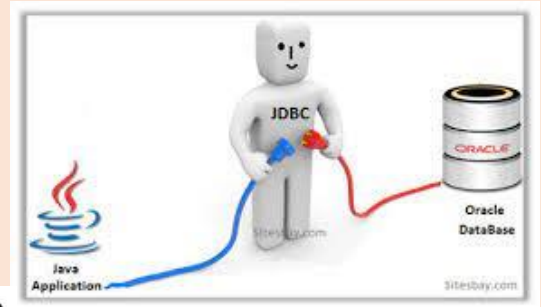
- **JDBC**

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.

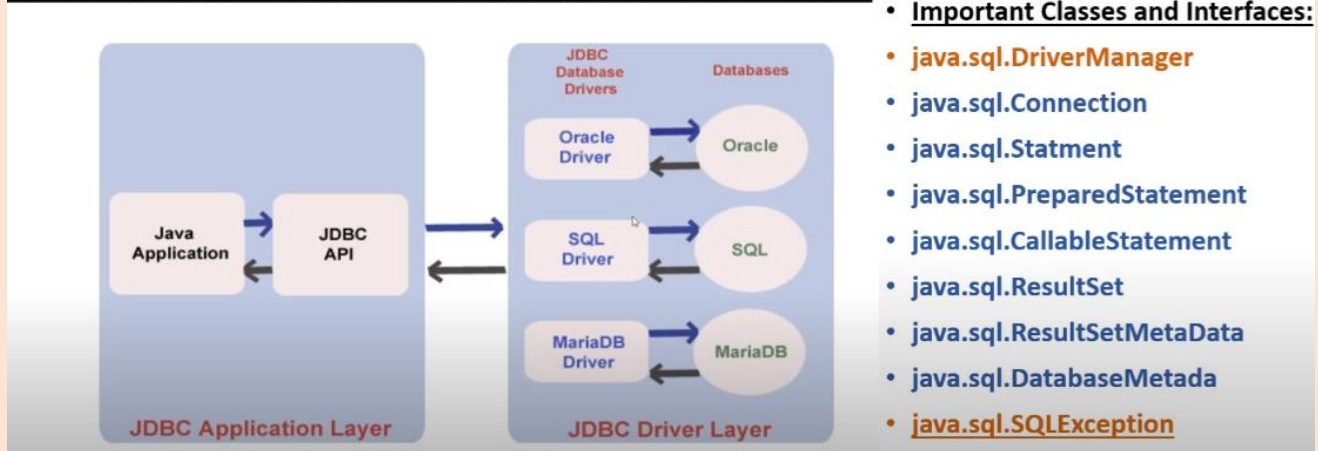
There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

- **JDBC API:-**



## **How JDBC Work(Architecture of JDBC)**



- **Important Classes and Interfaces:**

- **java.sql.DriverManager**
- **java.sql.Connection**
- **java.sql.Statement**
- **java.sql.PreparedStatement**
- **java.sql.CallableStatement**
- **java.sql.ResultSet**
- **java.sql.ResultSetMetaData**
- **java.sql.DatabaseMetada**
- **java.sql.SQLException**

A list of popular *classes* of JDBC API are given below:

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

- **Why Should We Use JDBC:-**

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But ODBC API uses ODBC driver which is written in C language (i.e., platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

- **What is API:-**

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

- **JDBC Driver:-**

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

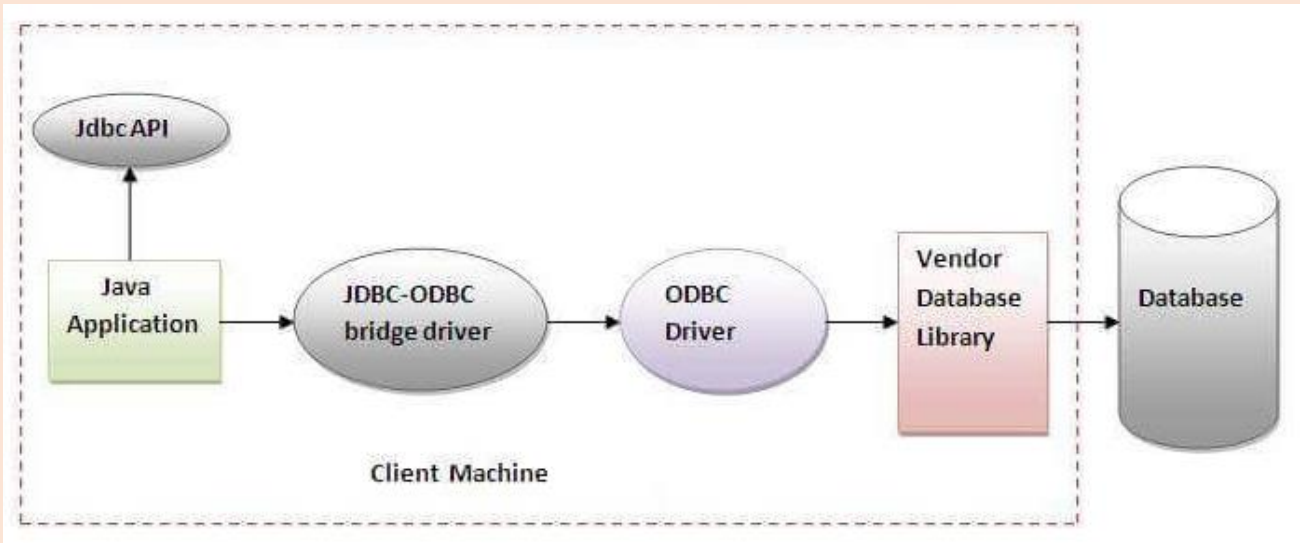
1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)

3. Network Protocol driver (fully java driver)

4. Thin driver (fully java 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.



*In Java 8, the JDBC-ODBC Bridge has been removed.*

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.

### Advantages:-

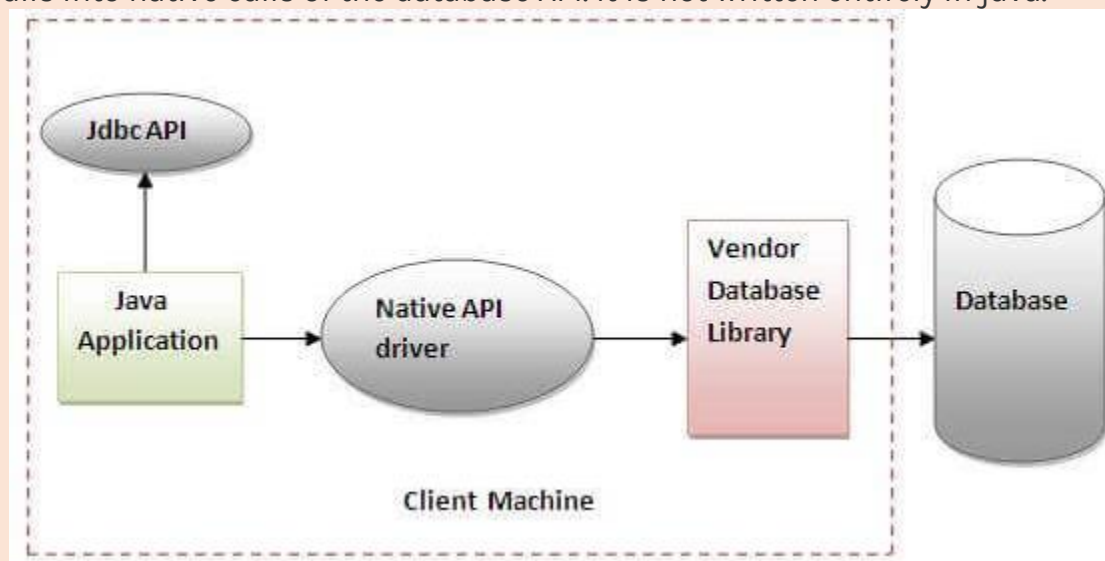
- easy to use.
- can be easily connected to any database.

### 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.



### Advantage:-

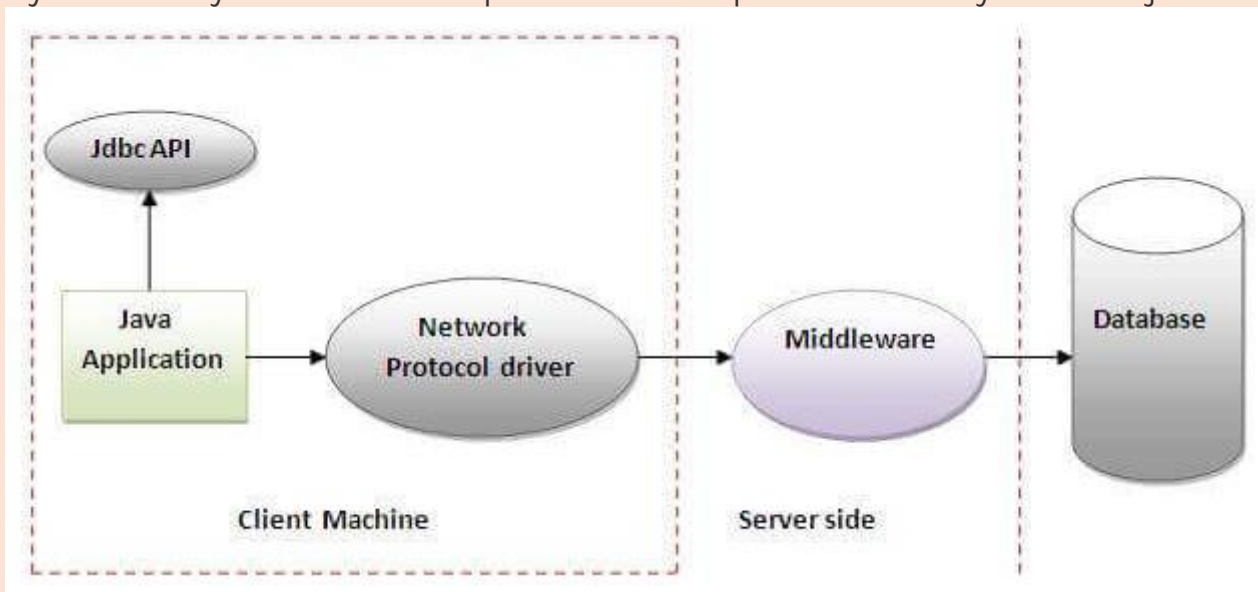
- performance upgraded than JDBC-ODBC bridge driver.

### 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.



#### **Advantage:-**

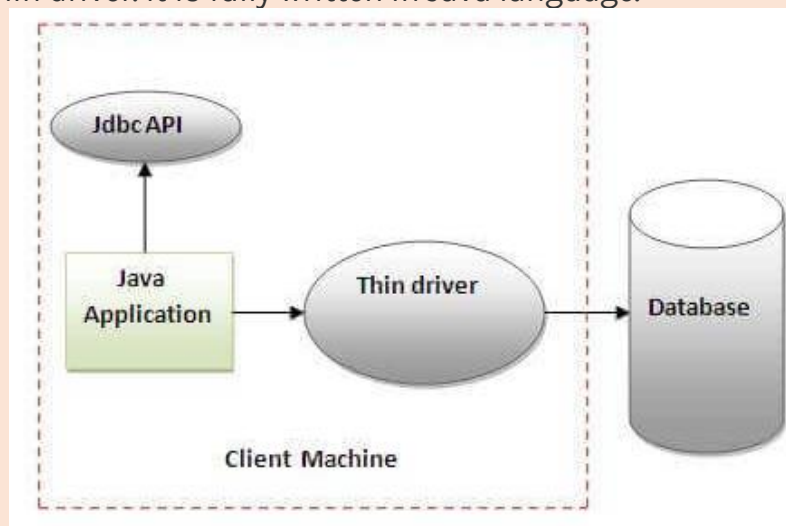
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

#### **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.



#### **Advantage:-**

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

#### **Disadvantage:-**

- Drivers depend on the Database.

### **• JDBC Connectivity Steps:-**

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:-

1. Import the package.

2. Load & register the driver.
3. Establish the connection .
4. Create the statement.
5. Execute the query.
6. Process result.
7. Close connection .

- **Steps to Connect with Database, explained in Detail (Java Database Connectivity) JDBC:-**

Steps to connect java program with database-

**1) load the driver:-**

`Class.forName("com.mysql.jdbc.Driver");//(inside of try-catch )`  
Or

`DriverManager.registerDriver(new com.mysql.jdbc.Driver());`

**2)Create a Connection:-**

`Connection con=DriverManager.getConnection("url","Username","Password");`  
Connection

`con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/dbname","root","root");`

**3)Create query, Statement , PreparedStatement , CallableStatement:-**

`String q="select * from students";`  
`Statement stmt=con.createStatement(); // stored in the stmt`  
`ResultSet set=stmt.executeQuery(q); // fire with the help of stmt`

**4)Process the data :-**

`while(set.next()) // its help to move into next row`  
{  
    `int id=set.getInt("studentID"); // we can also pass the column name`  
    `String name=set.getString("studentName");`  
    `System.out.println(id);`  
    `System.out.println(name);`  
}

**5) Close the connection:-**

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

**Remember Note:**

"If you want to fetch the data from the database you have to use **.executeQuery()** method and if want to update or perform the curd operation you have to use **.executeUpdate()** method."

- **MySQL-Connector Jar File:-**

- For using the database you have to download the MySQL jar file for java EE version. And built the Classpath of that jar file where you using the Java Project with MySQL.

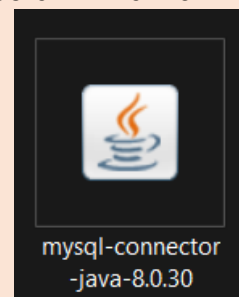
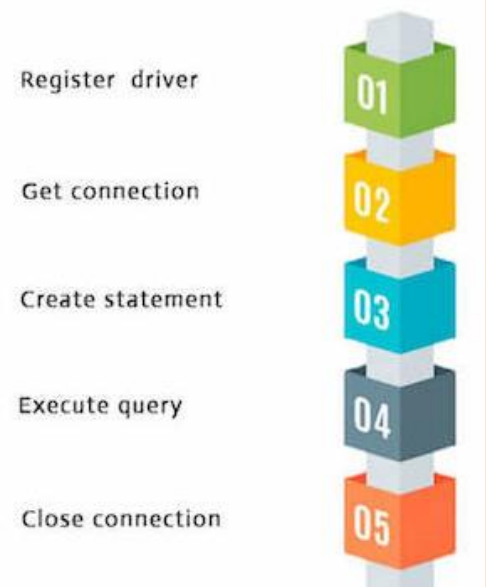
- Following steps To build the path of jar File in JAVA EE.
- I name my java-project as JDBC:-

- **Java Database Connectivity with MySQL:-**

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using **MySql** as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.





4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

### • Connection interface:-

- A Connection is a session between a Java application and a database. It helps to establish a connection with the database.
- The Connection interface is a factory of Statement, **PreparedStatement**, and **DatabaseMetaData**, i.e., an object of Connection can be used to get the object of Statement and **DatabaseMetaData**. The Connection interface provide many methods for transaction management like **commit()**, **rollback()**, **setAutoCommit()**, **setTransactionIsolation()**, etc.

*By default, connection commits the changes after executing queries.*

### • Commonly used methods of Connection interface:-

1) **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

2) **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) **public void setAutoCommit(boolean status):** is used to set the commit status. By default, it is true.

4) **public void commit():** saves the changes made since the previous commit/rollback is permanent.

5) **public void rollback():** Drops all changes made since the previous commit/rollback.

6) **public void close():** closes the connection and Releases a JDBC resources immediately.

### • Connection Interface Fields:-

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

- **TRANSACTION\_NONE:** No transaction is supported, and it is indicated by this constant.
- **TRANSACTION\_READ\_COMMITTED:** It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.
- **TRANSACTION\_READ\_UNCOMMITTED:** It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.
- **TRANSACTION\_REPEATABLE\_READ:** It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.
- **TRANSACTION\_SERIALIZABLE:** It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

### • Creating Connection with database (JDBC)

```
1 package JDBC;
2 // program for JDBC 1 program
3 import java.sql.Connection;
4
5 public class First_JDBC_making_connection {
6     public static void main(String[] args) throws ClassNotFoundException, SQLException {
7         // using try catch to overcome or get the exception
8         try{
9             // load the Driver
10             Class.forName("com.mysql.cj.jdbc.Driver");
11             // creating a connection
12             String url="jdbc:mysql://localhost:3306/jdbc";
13             String username="root";
14             String pass="uday123";
15             // use to get the connection
16             Connection con=DriverManager.getConnection(url,username,pass);
17             if(con.isClosed()) {
18                 System.out.println("Connection is still closed");
19             }
20             else {
21                 System.out.println("Connection is established");
22             }
23         }catch(Exception e) { e.printStackTrace(); } // catch the exception
24     }
25 }
```

Console:-

```
Console X
<terminated> First_JDBC_making_connection [Java Appli
Connection is established
```

## • Creating Table (CRUD):-

### • Create table Operation:-

```
First_JDBC_making_connection.java  JDBC_create_Table.java X
1 package JDBC;
2 //import java.sql.Connection;[]
5 import java.sql.*;
6 public class JDBC_create_Table {
7     public static void main(String[] args){
8         // using try catch to overcome or get the exception
9         try{
10            // load the Driver
11            Class.forName("com.mysql.cj.jdbc.Driver");
12            // creating a connection
13            String url="jdbc:mysql://localhost:3306/jdbc";
14            String username="root";
15            String pass="uday123";
16            // use to get the connection
17            Connection con=DriverManager.getConnection(url,username,pass);
18            // checking the connection
19            if(con.isClosed()) {
20                System.out.println("Connection is still closed"); }
21            else {
22                System.out.println("Connection is established"); }
23            // create a Query
24            String query="create table stu( id int(20) primary key auto_increment,Name varchar(100) not null ,city varchar(150)";
25            // create statement
26            Statement stmt=con.createStatement();
27            // Execute the statement by passing the query
28            stmt.executeUpdate(query);
29            System.out.println("Table created in database ");
30            // closing the connection
31            con.close();
32 }catch(Exception e) {e.printStackTrace();}
33 }
34 }
35 }
```

Database is create we can see in the **mysqlcommandline client**. Or we can in **MySQL Workbench**

```
mysql> show databases;
+-----+
| Database |
+-----+
| database_transaction |
| information_schema |
| jdbc |
| mysql |
| performance_schema |
| project |
| sys |
+-----+
7 rows in set (0.04 sec)

mysql> use jdbc;
Database changed
mysql> show tables;
+-----+
| Tables_in_jdbc |
+-----+
| stu |
+-----+
1 row in set (0.04 sec)
```

```
mysql> desc stu;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id | int | NO | PRI | NULL | auto_increment |
| Name | varchar(100) | NO | | NULL | |
| city | varchar(150) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.04 sec)
```

## • Inserting values into table (CRUD):-

```

1 package JDBC;
2 // this operation is an runtime polymorphism
3 import java.sql.*;
4 public class JDBC_inserting_data {
5     public static void main(String[] args) throws ClassNotFoundException , SQLException{
6         // TODO Auto-generated method stub
7         try {
8             // load the Driver
9             Class.forName("com.mysql.cj.jdbc.Driver");
10            // creating a connection
11            String url="jdbc:mysql://localhost:3306/jdbc";
12            String username="root";
13            String pass="uday123";
14            // use to get the connection
15            Connection con=DriverManager.getConnection(url,username,pass);
16            // checking the connection
17            if(con.isClosed()) {
18                System.out.println("Connection is still closed"); }
19            else {
20                System.out.println("Connection is established"); }
21            // creating a query
22            String q="insert into stu (name,city) values (?,?)";
23            // creating statement (Prepared statement)
24            PreparedStatement ps = con.prepareStatement(q);
25            // set the value of query
26            ps.setString(1,"Uday Sharma");
27            ps.setString(2,"Roorkee");
28            // now execute the query
29            ps.executeUpdate(); // there is no need to pass the query q again
30            // now we check the by convey a message that the data is inserted
31            System.out.println("the data is inserted...");
32            // closing the connection
33            con.close();
34            }catch(Exception e) { e.printStackTrace(); }
35        }
36    }

```

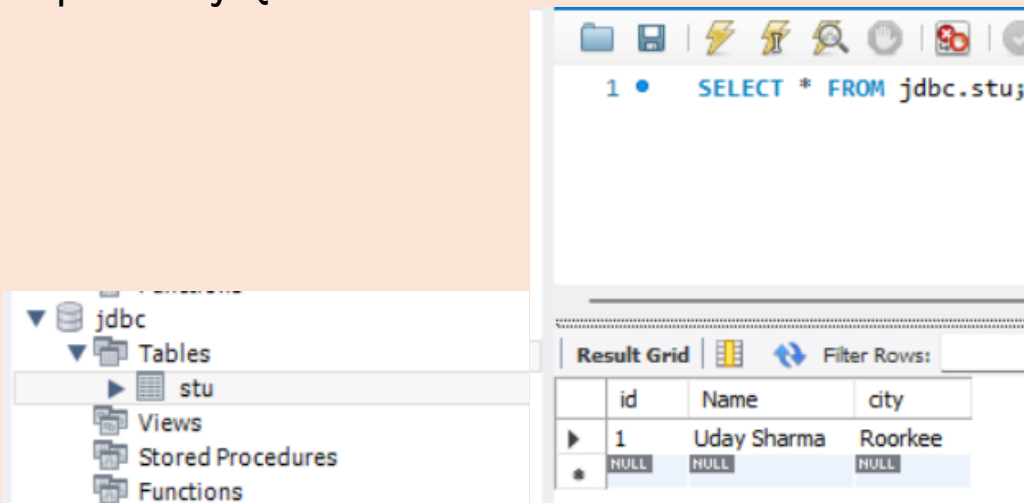
Output:- on Console

```

Console x
<terminated> JDBC_inserting_data [Java Application] C
Connection is established
the data is inserted...

```

Output:- on MySQL Workbench



The screenshot shows the MySQL Workbench interface. On the left, the 'Database Explorer' pane shows the 'jdbc' database with a table named 'stu'. The main window displays the SQL query 'SELECT \* FROM jdbc.stu;' and the 'Result Grid' tab. The result grid shows the following data:

	id	Name	city
1	1	Uday Sharma	Roorkee
*	NULL	NULL	NULL



- Dynamic Values inserted;

- Inserting values into table by user input using BufferedReader class (CRUD):-

```
package JDBC;
import java.io.*;
import java.sql.*;
public class Inserting_data_dyanamic_input {
    public static void main(String[] args) {
        try {
            // load the Driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // creating a connection
            String url="jdbc:mysql://localhost:3306/jdbc";
            String username="root";
            String pass="uday123";
            // use to get the connection
            Connection con=DriverManager.getConnection(url,username,pass);
            // checking the connection
            if(con.isClosed()) {
                System.out.println("Connection is still closed"); }
            else {
                System.out.println("Connection is established"); }
            String q="insert into stu(name,city) values (?,?)";
            // prepared statement
            PreparedStatement pstmt= con.prepareStatement(q);
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            // Taking user input
            System.out.print("Enter your name : ");
            String name=br.readLine();
            System.out.println("Enter the city : ");
            String city=br.readLine();
            // now execute the query
            pstmt.executeUpdate();
            // now we check the by convey a message that the data is inserted
            System.out.println("the data is inserted...");
            // closing the connection
            con.close();
        }catch(Exception e) { e.printStackTrace(); }
    }
}
```

Output:- On Console:-

```
Connection is established
Enter your name : Avish Tyagi
Enter the city :
Manglore
the data is inserted...
```

Output:- MySQL CommandLine :-

```
mysql> select * from stu;
+----+-----+-----+
| id | Name       | city    |
+----+-----+-----+
| 1  | Uday Sharma | Roorkee |
| 2  | Anshul Lakher | Kashipur |
| 3  | Avish Tyagi  | Manglore |
+----+-----+-----+
3 rows in set (0.00 sec)
```

- Updating data using the dynamic values using BufferedReader:-

```
package JDBC;
import java.io.*;
import java.sql.*;
public class Update_IN_JDBC {
    public static void main(String[] args) {
        try {
            // load the Driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // creating a connection
            String url="jdbc:mysql://localhost:3306/jdbc";
            String username="root";
            String pass="uday123";
            // use to get the connection
            Connection con=DriverManager.getConnection(url,username,pass);
            // checking the connection
            if(con.isClosed()) {
                System.out.println("Connection is still closed"); }
            else {
                System.out.println("Connection is established"); }
            String q="update stu set name=?,city=? where id=? ";
            // prepared statement
            PreparedStatement pstmt= con.prepareStatement(q);
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            // Taking user input of new data from the input
            System.out.print("Enter your new name : ");
            String name=br.readLine();
            System.out.println("Enter your new city : ");
            String city=br.readLine();
            System.out.println("Enter the student id where you want to update the data : ");
            // convert string into integer using parseInt
            int id=Integer.parseInt(br.readLine());
            // set the value of query
            pstmt.setString(1,name);
            pstmt.setString(2,city);
            pstmt.setInt(3,id);
            // now execute the query
            pstmt.executeUpdate();
            // now we check the by convey a message that the data is inserted
            System.out.println("the data is updated...");
            // closing the connection
            con.close();
        }catch(Exception e) { e.printStackTrace(); }
    }
}
```

**Output:-**

Before Update Query

```
mysql> select * from stu;
+----+-----+-----+
| id | Name      | city      |
+----+-----+-----+
| 1  | Uday Sharma | Roorkee   |
| 2  | Anshul Lakher | Kashipur |
| 3  | Avish Tyagi  | Manglore  |
| 4  | Yashas Gupta | Jawalapur |
+----+-----+-----+
4 rows in set (0.00 sec)
```

After Update Query

```
mysql> select * from stu;
+----+-----+-----+
| id | Name      | city      |
+----+-----+-----+
| 1  | Uday Sharma | Roorkee   |
| 2  | Anshul Lakher | Kashipur |
| 3  | Yash Kapoor  | Saharanpur |
| 4  | Yashas Gupta | Jawalapur |
+----+-----+-----+
4 rows in set (0.00 sec)
```

## • Fetching details from database (CRUD):-

```
*JDBC_Fetching_Data_MySQL.java × First_JDBC_making_connection.java
1 package JDBC;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 public class JDBC_Fetching_Data_MySQL {
8     public static void main(String[] args) throws ClassNotFoundException, SQLException {
9         // Connect to the MySQL Database
10        try {
11            Class.forName("com.mysql.cj.jdbc.Driver");
12            String url="jdbc:mysql://localhost:3306/jdbc";
13            String username="root";
14            String pass="uday123";
15            Connection con=DriverManager.getConnection(url,username,pass);
16            // Checking the Connection
17            if(con.isClosed()) {
18                System.out.println("Connection is still closed");
19            }
20            else {
21                System.out.println("Connection is established");
22            }
23            //Generate the Query
24            String query="select * from stu;";
25            // Prepared the Query
26            PreparedStatement pstmt=con.prepareStatement(query);
27            // Execute the Query
28            ResultSet rs=pstmt.executeQuery(query);
29            // Fetching the data from database JDBC and the Table Stu
30            while(rs.next()) {
31                System.out.println("ID :"+rs.getInt(1));
32                System.out.println("Name :"+rs.getString(2));
33                System.out.println("City :"+rs.getString(3));
34            }
35            System.out.println("Data is Shown above ");
36            con.close();
37        } catch (Exception e) { e.printStackTrace(); }
38    }
39 }
```

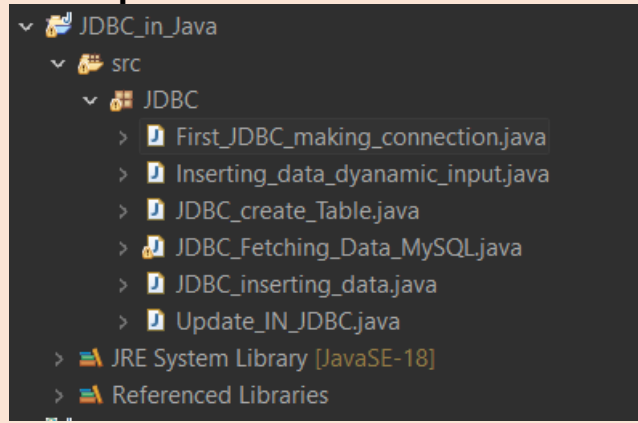
### Output:-

```
Console ×
<terminated> JDBC_Fetching_Data_MySQL [Java
Connection is established
ID :1
Name :Uday Sharma
City :Roorkee
ID :2
Name :Anshul Lakher
City :Kashipur
ID :3
Name :Yash Kapoor
City :Saharanpur
ID :4
Name :Yashas Gupta
City :Jawalapur
Data is Shown above
```

## • Data Access Object in JDBC - (DAO classes)

DAO is an abstraction for accessing data, the idea is to separate the technical details of data access from the rest of the application. It can apply to any kind of data. JDBC is an API for accessing relational databases using Java.



- Topics Covered in the NOTES :-



Eclipse IDE , JAVA EE , JDK-18



➤ Go check out my [LinkedIn profile](#) for more notes and other resources content

@Uday Sharma  
mrudaysharma4600@gmail.com

<https://www.linkedin.com/in/uday-sharma-602b33267>