JDBC

Java Database Connectivity

JDBC

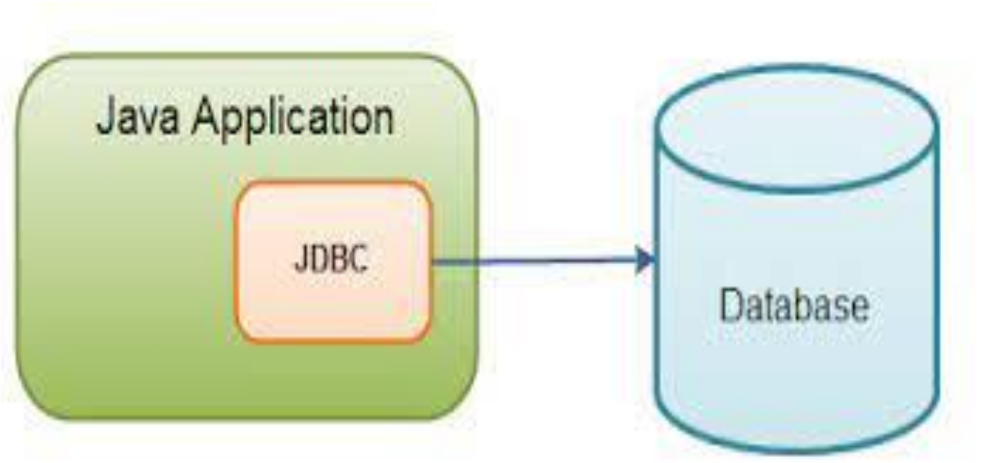
JDBC is **API** and It is used to Establish the Connection between Java Program and Database.

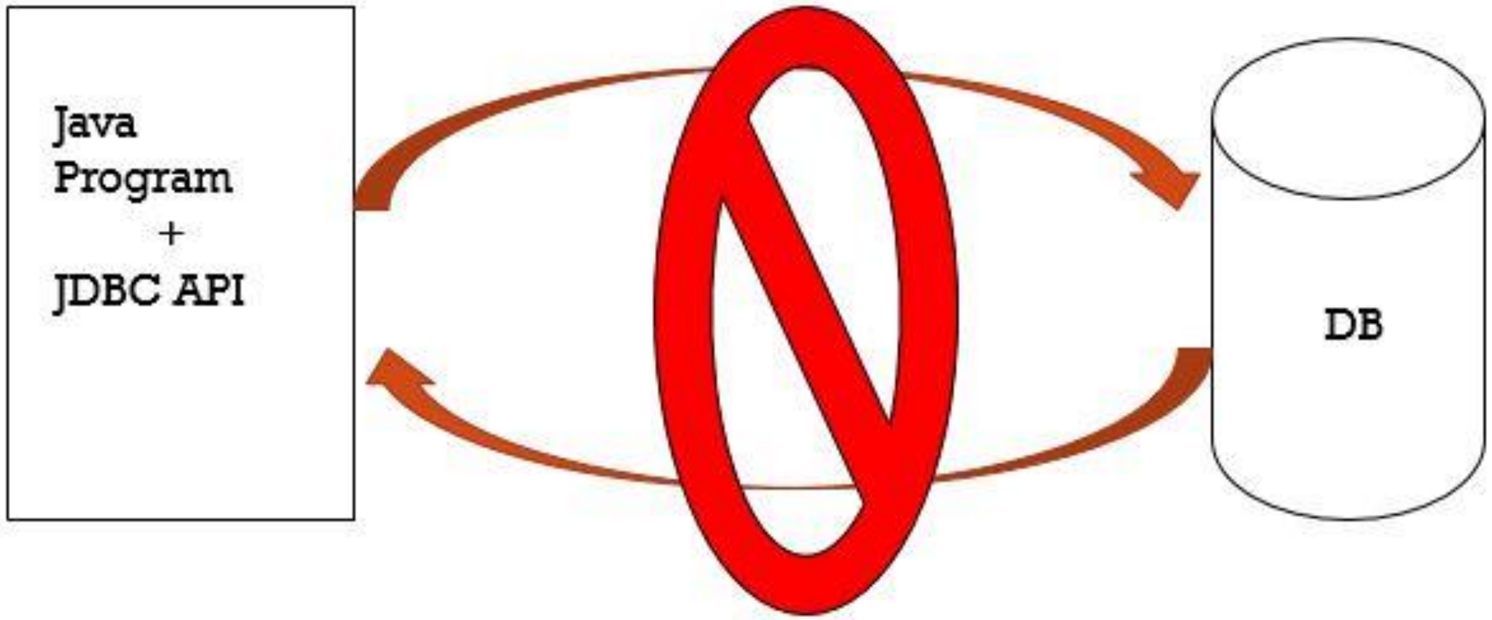
What is API ?

API is a software which is allows two applications to communicate with each other.

We do not install JDBC explicitly because After JDK 1.0 version JDBC is present inside a JDK only.

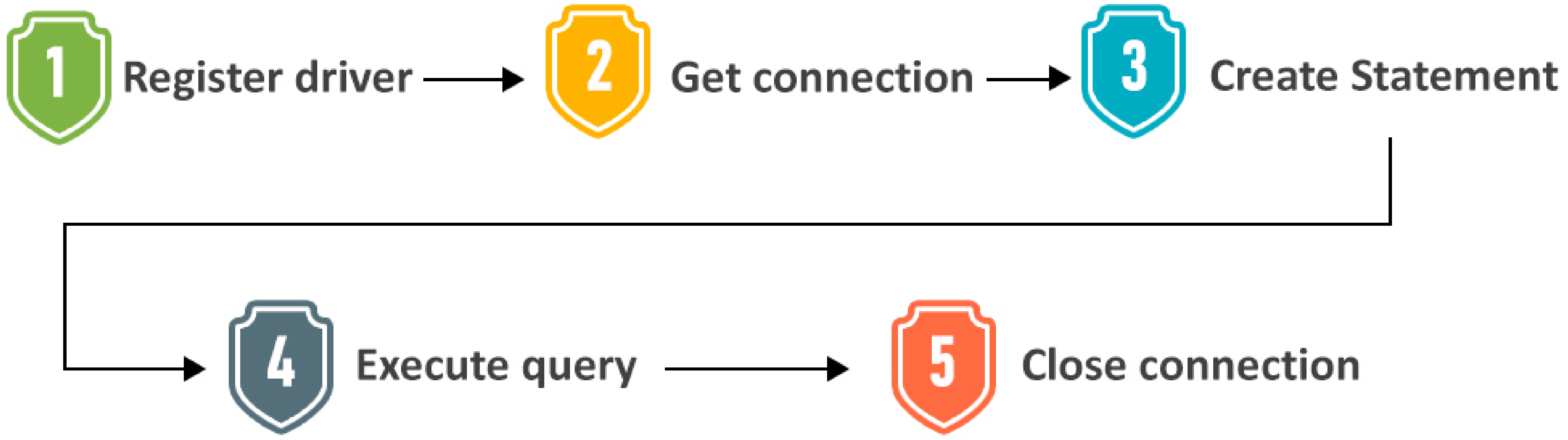
JDBC is the one and only API which helps us to communicate with Database.





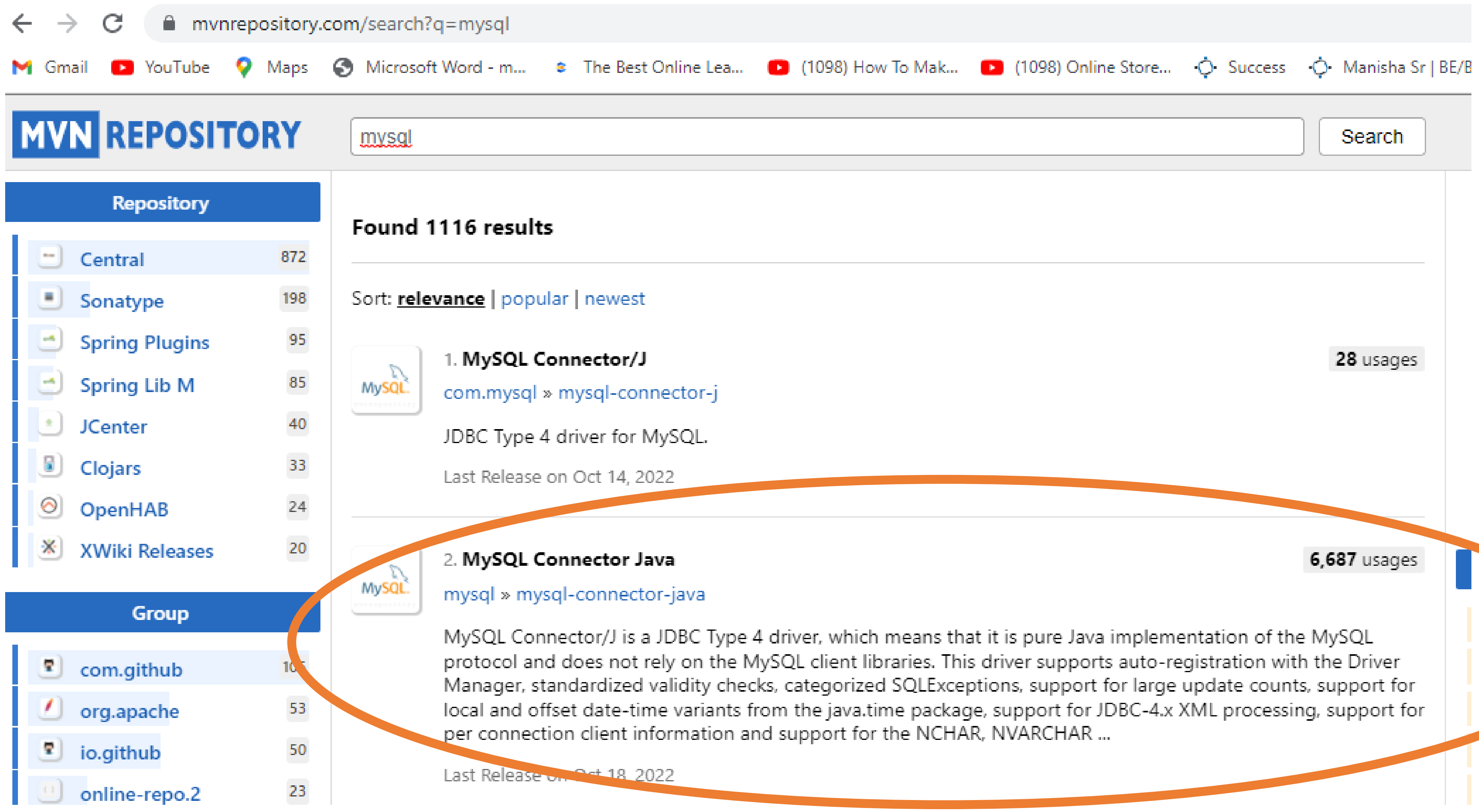
JDBC can’t talk with Database directly

Java Database Connectivity



How to download MySQL Connector/Driver

From MVN Repository -> Search MySQL-connector-java



mvnrepository.com/search?q=mysql

Repository	
Central	872
Sonatype	198
Spring Plugins	95
Spring Lib M	85
JCenter	40
Clojars	33
OpenHAB	24
XWiki Releases	20

Group	
com.github	10
org.apache	53
io.github	50
online-repo.2	23

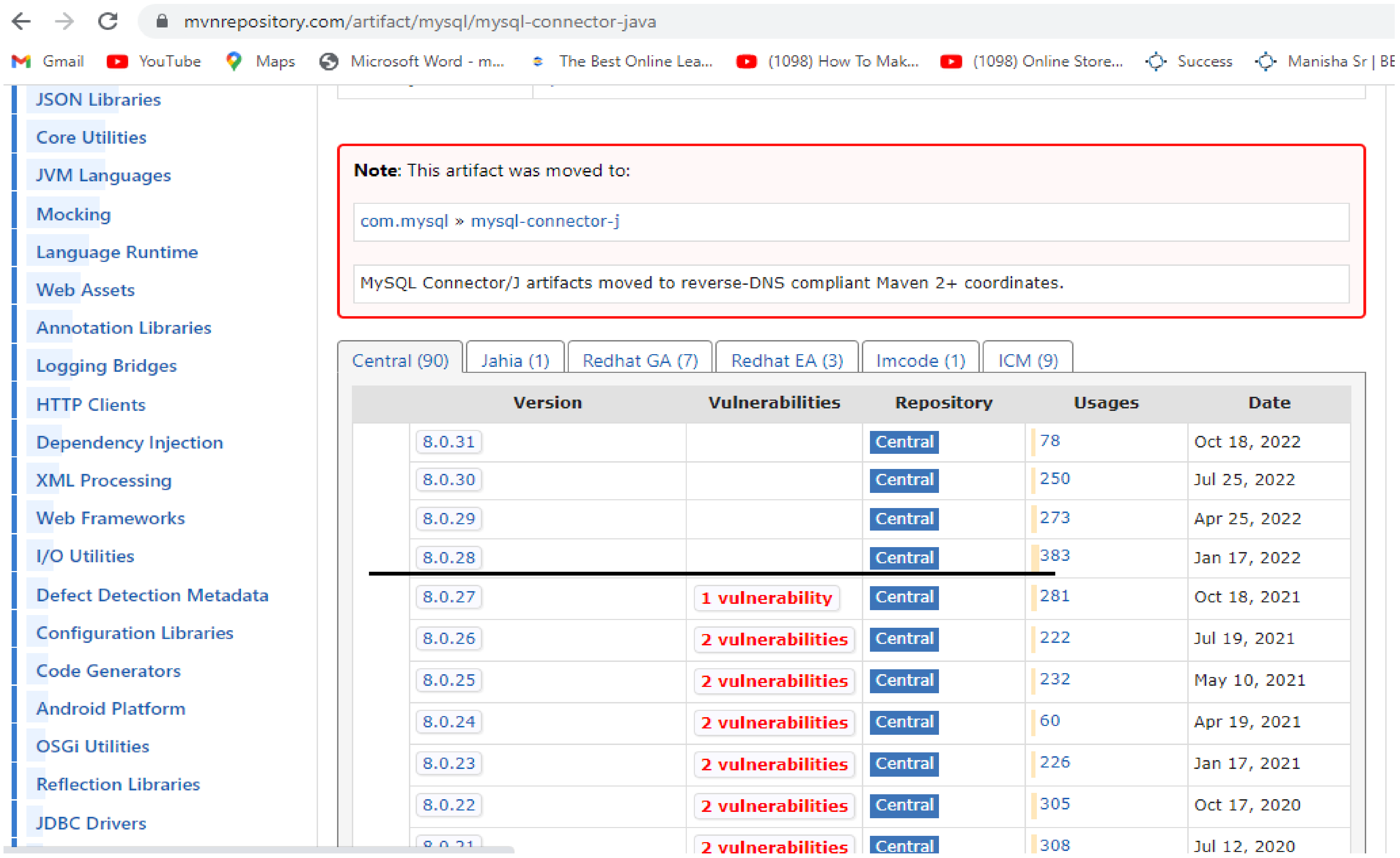
Found 1116 results

Sort: **relevance** | popular | newest

1. **MySQL Connector/J** — **28** usages
com.mysql » mysql-connector-j
JDBC Type 4 driver for MySQL.
Last Release on Oct 14, 2022

2. **MySQL Connector Java** — **6,687** usages
mysql » mysql-connector-java
MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...
Last Release on Oct 18, 2022

Choose version which is most used
For Example Version 8.0.28 -> Usages 383


mvnrepository.com/artifact/mysql/mysql-connector-java

JSON Libraries
Core Utilities
JVM Languages
Mocking
Language Runtime
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Dependency Injection
XML Processing
Web Frameworks
I/O Utilities
Defect Detection Metadata
Configuration Libraries
Code Generators
Android Platform
OSGi Utilities
Reflection Libraries
JDBC Drivers

Note: This artifact was moved to:

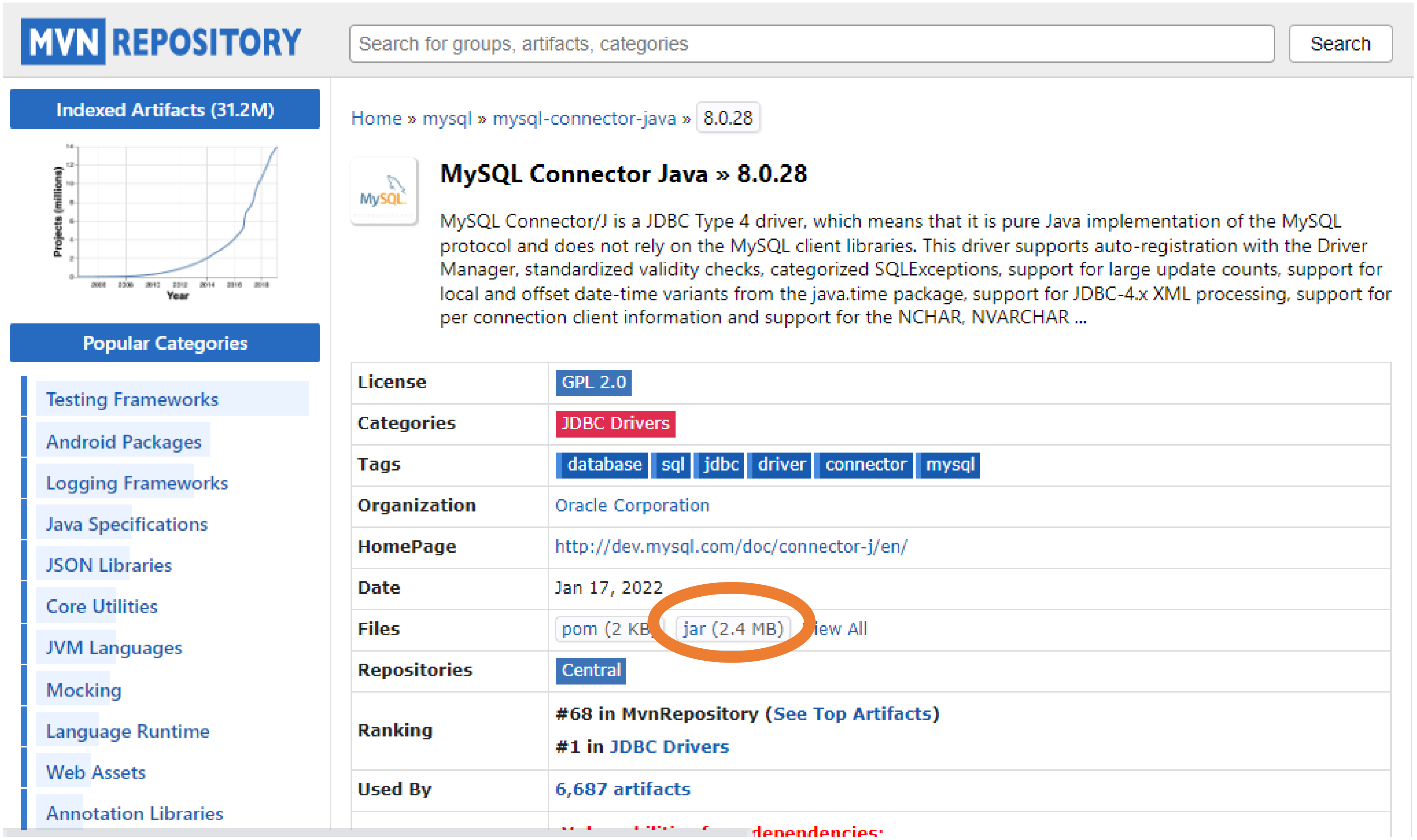
com.mysql » mysql-connector-j

MySQL Connector/J artifacts moved to reverse-DNS compliant Maven 2+ coordinates.

Central (90) | Jahia (1) | Redhat GA (7) | Redhat EA (3) | Imcode (1) | ICM (9)

Version	Vulnerabilities	Repository	Usages	Date
8.0.31		Central	78	Oct 18, 2022
8.0.30		Central	250	Jul 25, 2022
8.0.29		Central	273	Apr 25, 2022
8.0.28		Central	383	Jan 17, 2022
8.0.27	1 vulnerability	Central	281	Oct 18, 2021
8.0.26	2 vulnerabilities	Central	222	Jul 19, 2021
8.0.25	2 vulnerabilities	Central	232	May 10, 2021
8.0.24	2 vulnerabilities	Central	60	Apr 19, 2021
8.0.23	2 vulnerabilities	Central	226	Jan 17, 2021
8.0.22	2 vulnerabilities	Central	305	Oct 17, 2020
8.0.21	2 vulnerabilities	Central	308	Jul 12, 2020

Download the jar



MVN REPOSITORY

Search for groups, artifacts, categories

Search

Indexed Artifacts (31.2M)

Popular Categories

- Testing Frameworks
- Android Packages
- Logging Frameworks
- Java Specifications
- JSON Libraries
- Core Utilities
- JVM Languages
- Mocking
- Language Runtime
- Web Assets
- Annotation Libraries

Home » mysql » mysql-connector-java » 8.0.28

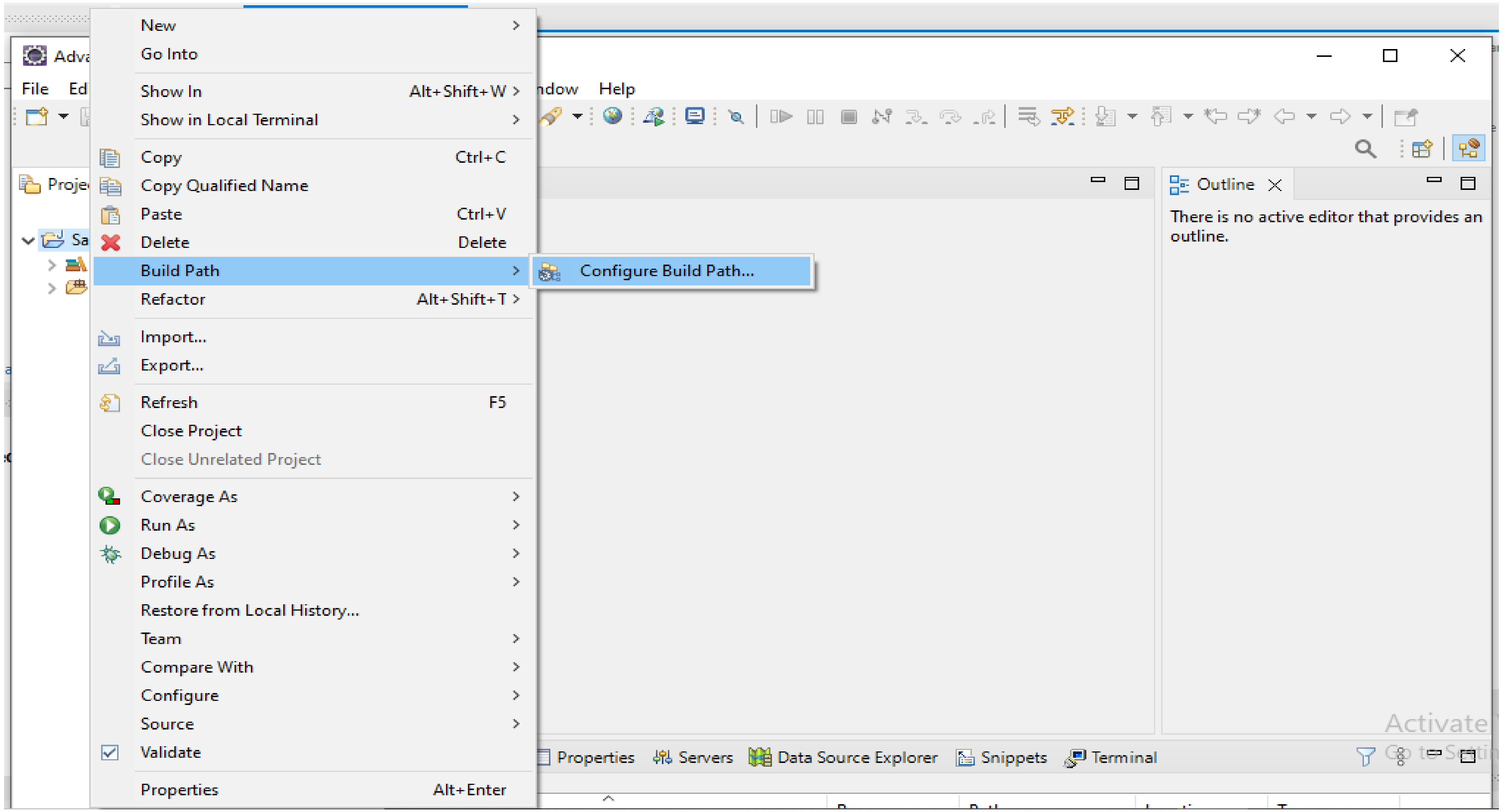
MySQL Connector Java » 8.0.28

MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...

License	GPL 2.0
Categories	JDBC Drivers
Tags	database sql jdbc driver connector mysql
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	Jan 17, 2022
Files	pom (2 KB) jar (2.4 MB) View All
Repositories	Central
Ranking	#68 in MvnRepository (See Top Artifacts) #1 in JDBC Drivers
Used By	6,687 artifacts

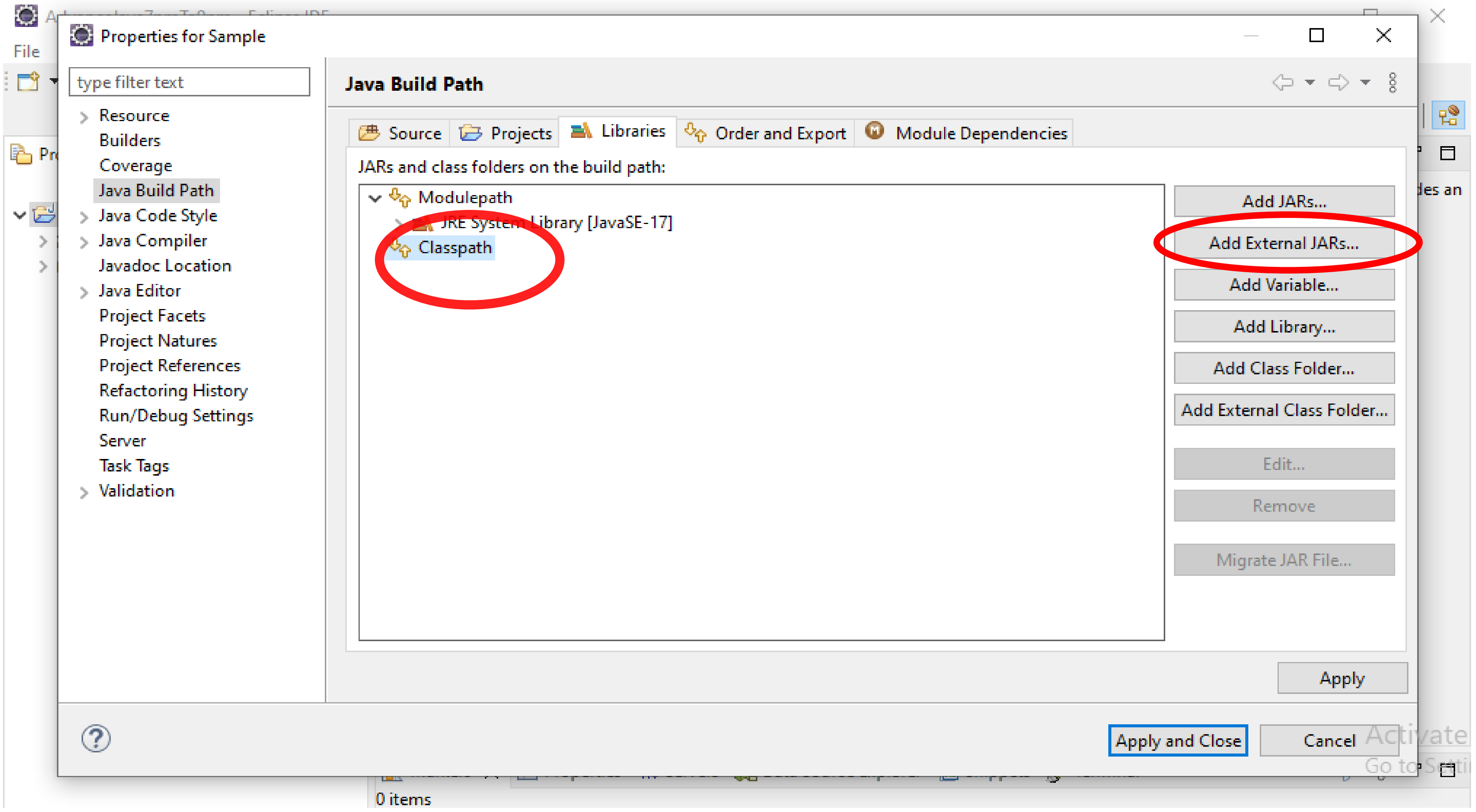
How to add MySQL Driver into Project

Right Click on Project -> Select Build Path -> Select Configure Build Path

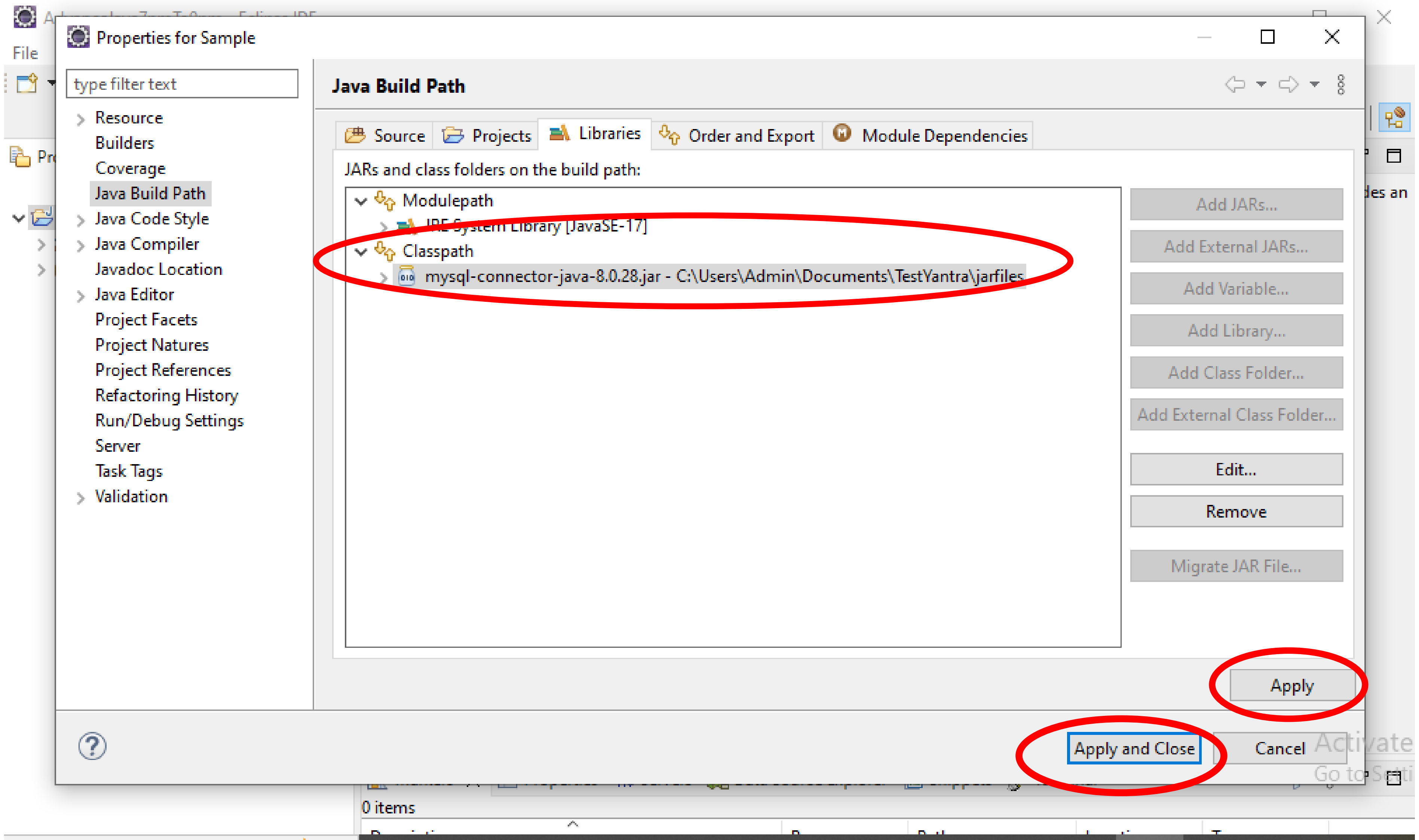


There is no active editor that provides an outline.

In Java Build Path -> Goto Libraries -> Select ClassPath -> Add External JARs



After adding MySQL Driver -> Click on Apply -> Apply and Close

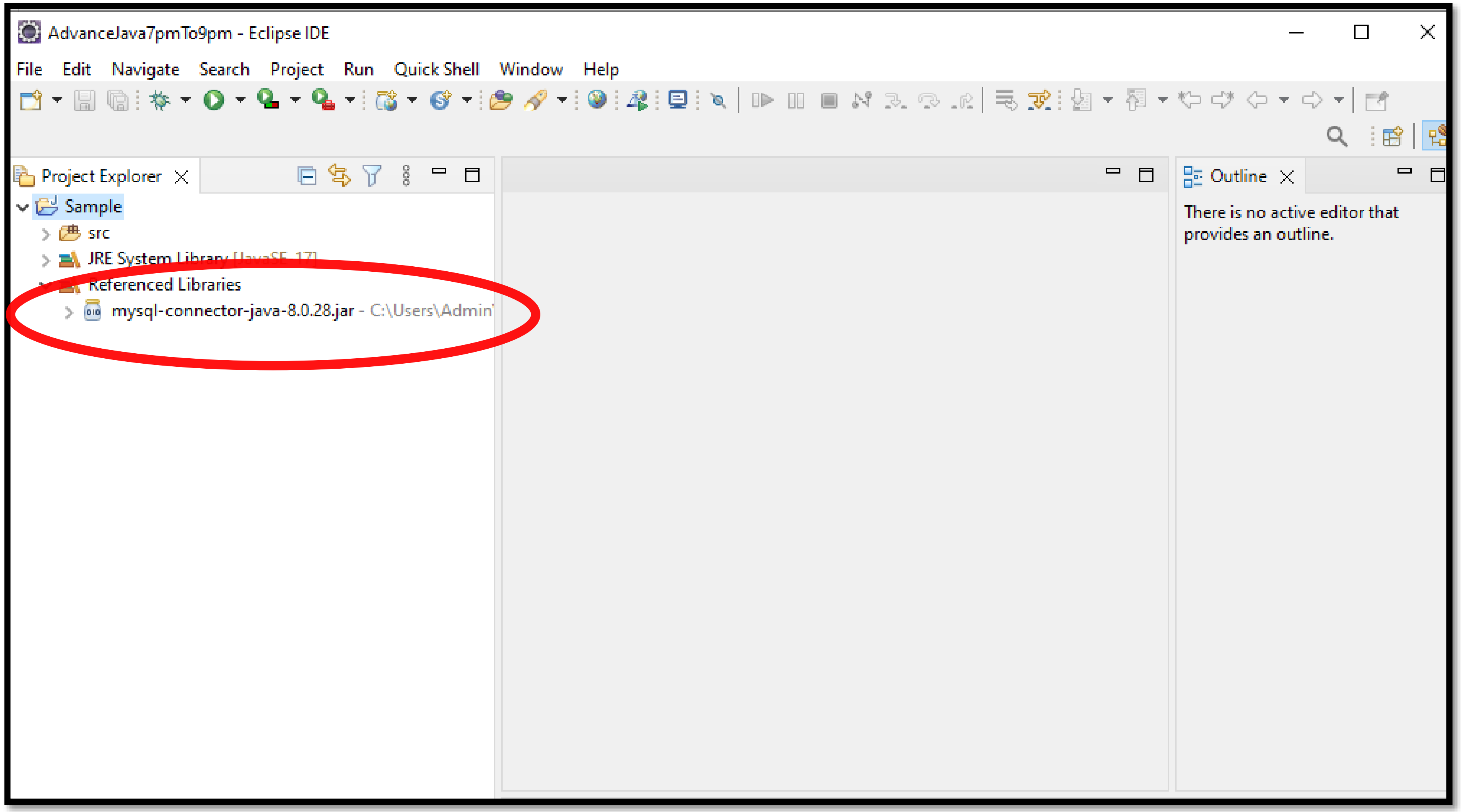

Properties for Sample

Java Build Path

JARs and class folders on the build path:

Modulepath
JRE System Library [JavaSE-17]
Classpath
mysql-connector-java-8.0.28.jar - C:\Users\Admin\Documents\TestYantra\jarfiles

mysql-connector (driver) added into project


AdvanceJava7pmTo9pm - Eclipse IDE
File Edit Navigate Search Project Run Quick Shell Window Help
Project Explorer
Sample
src
JRE System Library [JavaSE-17]
Referenced Libraries
mysql-connector-java-8.0.28.jar - C:\Users\Admin\
Outline

There is no active editor that provides an outline.

Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class

forName() is a Public and Static Method and its Accept only one String Parameter.

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

Create the connection object

The **getConnection()** method of Driver Manager class is used to establish connection with the database.

```
DriverManager.getConnection(“url", “user name", "password");
```

Create the Statement

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

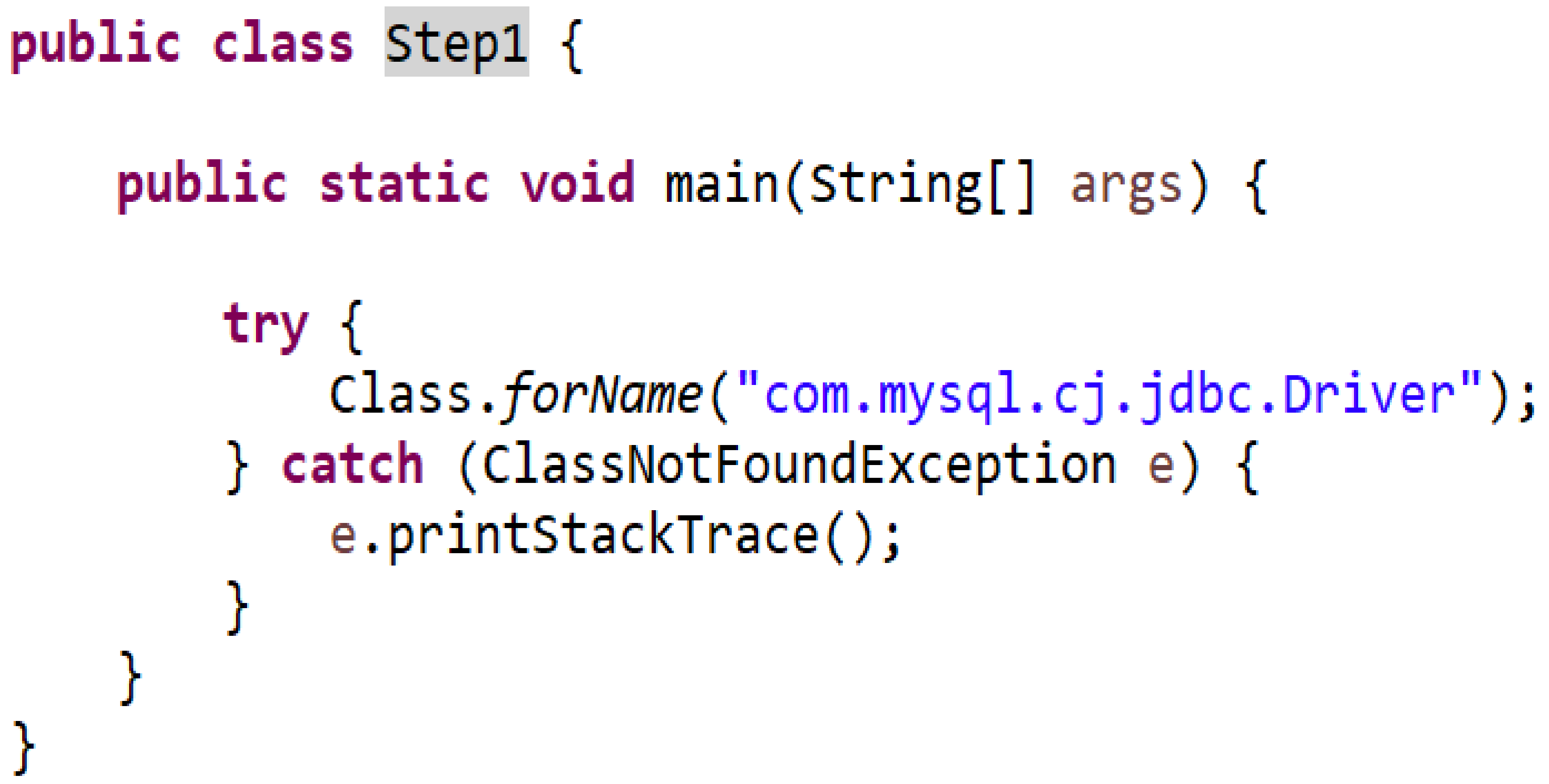
```
Statement stmt=con.createStatement();
```

Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database.

```
String query = “INSERT INTO STUDENT VALUES(1, ‘Priya’, 20);
stmt.execute(query);
```

Step 1: Load Driver



```
public class Step1 {

    public static void main(String[] args) {

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Step 2: Get Connection

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

public class Step2 {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
```

Step 3: Create Statement

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

public class Step3 {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Step 4: Execute Query

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

public class Step4 {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
            String query = "insert into jdbc_table.student values(11,'dimple',34.56)";
            stmt.execute(query);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Step 5: Close Connection

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

public class Step5 {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
            String query = "insert into jdbc_table.student values(11,'dimple',34.56)";
            stmt.execute(query);
            con.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }}
}
```

Insert Data

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

public class Step5 {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
            String query = "insert into jdbc_table.student values(11,'dimple',34.56)";
            stmt.execute(query);
            con.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }}
}
```

Update Data

```
public class UpadteData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
            String query = "update jdbc_tables.student set name = 'raj dimple' where id = 11";
            stmt.execute(query);
            con.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }}
```

Delete Data

```java
public class UpadteData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url,username,password);
            Statement stmt = con.createStatement();
            String query = "delete from jdbc_tables.student where id = 11";
            stmt.execute(query);
            con.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }}
```

Assignment - 01

Create a table : Table Name -> employee
Columns -> id , name , designation , salary

Using 5 steps perform all CRUD Operations -> Insert,Update,Delete,Select

Select Data

```
public class PrintData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, username, password);
            Statement stmt = con.createStatement();
            String query = "Select * from jdbc_tables.student";
            ResultSet rs = stmt.executeQuery(query);
            while(rs.next()) {
                System.out.println(rs.getInt(1));
                System.out.println(rs.getString(2));
                System.out.println(rs.getDouble(3));
                System.out.println("---------------------------");
            }
            con.close();

        } catch (ClassNotFoundException | SQLException e) {

            e.printStackTrace();
        }
    }
}
```

```java
public class PrintDataById {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306";
        String username = "root";
        String password = "root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, username, password);
            Statement stmt = con.createStatement();
            String query = "Select * from jdbc_tables.student where id = 13";
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                System.out.println(rs.getInt(1));
                System.out.println(rs.getString(2));
                System.out.println(rs.getDouble(3));
            }
            con.close();
        } catch (ClassNotFoundException | SQLException e) {

            e.printStackTrace();
        }
    }
}
```

Create Driver Object :

```
Driver driver = new Driver();
DriverManager.registerDriver(driver);
```

- Driver is class and its present in com.mysql.cj.jdbc package
- DriverManager is class and its present java.sql package.
- registerDriver() is a Public and Static Method and its Accept only one Object Parameter

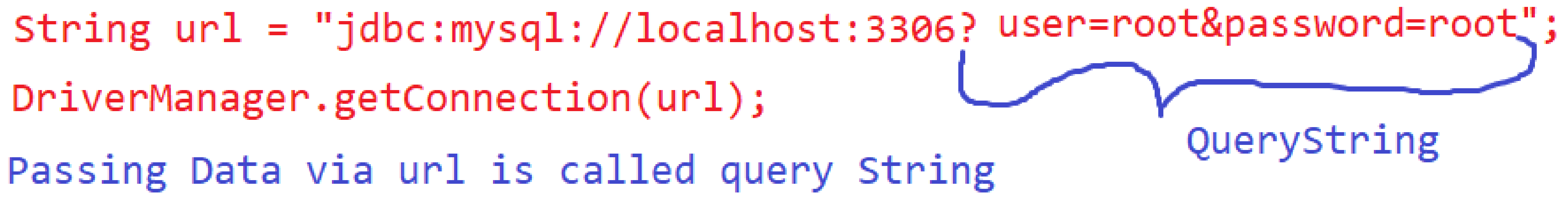
Establish the Connection

In Establish Connection step Connect the java application to DataBase through the jdbc.

For Establish the Connection Mainly in Three ways .

- DriverManager.getConnection(String,String,String) : Connection
- DriverManager. getConnection(String) : Connection

Connection : Connection is a interface and it is present in a java.sql package



```
String url = "jdbc:mysql://localhost:3306? user=root&password=root";
DriverManager.getConnection(url);
```

Passing Data via url is called query String

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import com.mysql.cj.jdbc.Driver;
public class Step2 {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306?user=root&password=root";
        try {
            // Step 1
            Driver d = new Driver();
            DriverManager.registerDriver(d);
            // Step 2
            Connection con = DriverManager.getConnection(url);
            // Step 3
            Statement stmt = con.createStatement();
            String query = "insert into jdbc_tables.student values(14,'raj',67.89)";
            //Step 4
            stmt.execute(query);
            //Step 5
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Establish the Statement

- The statement interface Statements are used to send SQL commands to the database and receive data from the database. in Java it provides methods to execute queries with the database.
- There are different types of statements that are used in JDBC as follows:
    - Statement
    - Prepared Statement
    - Callable Statement

Prepared Statement

- It is interface extends from the statement interface and it is present inside java.sql package.
- When we execute dynamic query ,on that we move on to Prepared Statement but in a Statement also we execute dynamic query but that is not good practise of coding

Placeholder or Delimiter (?) :

A placeholder expression provides a location in a SQL statement for which a third-generation language bind variable will provide a value

PreparedStatement pstmt = con.preparedStatement(String);

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
public class InsertData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306?user=root&password=root";
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url);
            String query = "insert into jdbc_tables.student values(?,?,?)";
            PreparedStatement pstmt = con.prepareStatement(query);
            pstmt.setInt(1, 15);
            pstmt.setString(2, "priya");
            pstmt.setDouble(3, 34.567);
            pstmt.execute();
            con.close();
            System.out.println("Data inserted");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

What is Connection?

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 etc., i.e., an object of Connection can be used to get the object of Statement etc.

What is DriverManager?

The DriverManager class is the component of JDBC API and also a member of the java.sql package.
It keeps track of the drivers that are available and handles establishing a connection between a database and appropriate driver.

What is Statement?

The Statement is an interface which provides methods to execute queries with the database. The statement interface is a factory of ResultSet ie., it provides factory method to get the object of ResultSet.

What is Prepared Statement?

The Prepared Statement is a sub-interface of Statement. It is used to execute parameterized query.
Example: String sql = “insert into EMP values (?,?,?)

What is the difference between Statement and Prepared Statement?

Statement	Prepared Statement
It is used for Static Query	It is used for Dynamic Query
It cannot accept parameter at run time	It can accept input parameter at runtime
It is a base interface	It extends Statement interface
It is slow as compared to Prepared Statement	It is faster because it is used for executing pre-compiled SQL statement
It is suitable for executing DDL commands.	It is suitable for executing DML commands.

EXECUTE QUERY

- We can execute the query in JDBC by using three methods which belongs to Statement Interfaces , based on requirement we choose one of them to execute the query.
- Three execute query methods in JDBC listed below :
  1. execute()
  2. executeUpdate()
  3. executeQuery()
- Arguments need to pass will differ from one statement to other for all three methods.
- For Statement we have to pass the SQL query as argument , for PreparedStatement no need to pass SQL query.

1. **execute()**

- **This method is used for execution of all kinds of SQL statements.**
  (select , insert ,update & delete)
- **Its return type is Boolean**
  - **If query return ResultSet object then it returns true .**
  - **If query returns int values or nothing it returns false .**

Example(Statement) :

```
Statement statement = connection.createStatement() ;
statement. execute(SQL);
```

Example(PreparedStatement) :

```
PreparedStatement preparedStatement = connection.prepareStatement(SQL);
preparedStatement.execute();
```

2. executeUpdate()

- This method is used for execution of SQL statements which updates or modifies the database table .
- this method returns int value which represents the number of rows affected by the executed query .
- If there is no any modifications took place in the targeted table then it returns 0 .
- This method is used for non-select queries

Example :

DML -> Insert , Update and Delete

Example(Statement) :

```
Statement statement = connection.createStatement() ;
statement.executeUpdate(SQL);
```

Example(PreparedStatement) :

```
PreparedStatement preparedStatement = connection.prepareStatement(SQL);
preparedStatement.executeUpdate();
```

3. executeQuery()

- This method is used for execution of SQL statements which retrieves the data from the database .
- this method returns ResultSet object which contains result table returned by the query.
- This method is used to execute only select queries

Example :

DQL : Select

Example(Statement) :

```
Statement statement = connection.createStatement() ;
statement.executeQuery(SQL);
```

Example(PreparedStatement) :

```
PreparedStatement preparedStatement = connection.prepareStatement(SQL);
preparedStatement.executeQuery();
```

CLOSE CONNECTION

- **Close() non static method of Connection interface used to close the Connection between. the JAVA application and the database.**
- **Connection is a costly resource which will effect performance of the application if connection is not closed.**
- **This method throws SQL exception which is checked exception .**

Example(Statement) :

```
connection.close() ;
```

.Explain execute (), executeQuery (), executeUpdate ()?

➔ **execute()**

It executes the given query and returns Boolean value.

If it is used to execute SELECT QUERY -> it returns true.

If it is used to execute NON – SELECT QUERY -> it returns false.

➔ **executeQuery()**

It executes the given query and returns ResultSet

It is used to execute SELECT Query. It returns the object of ResultSet.

If we use executeQuery () for NON-SELECT QUERY, it will throw SQL Exception.

➔ **executeUpdate ()**

It executes the given query and returns int value.

Int value which represents number of rows affected.

It is used to execute Non-Select query.

If we use for Select Query, it will throw SQL Exception.

What is ResultSet?

The SQL Statements that read data from a database query, return the data in a result set.

The SELECT statement is the standard way to select rows from a database and view them in a result set.

The **java.sql.ResultSet** interface represents the result set of a database query.

The ResultSet object maintains a cursor that points to the current row in the result set.

.Difference between execute (), executeQuery (), executeUpdate ()?

execute ()	**executeQuery ()**	**executeUpdate ()**
It returns Boolean value	It returns ResultSet	It returns int value
It is used to execute all SQL Queries	It is used to execute Select Query	It is used to execute Non-Select Query
It returns true if we use Select Query, returns false if we use Non-Select Query	If it is used to execute Non-Select Query, it throws SQL Exception.	If it is used to execute Select Query, it throws SQL Exception.