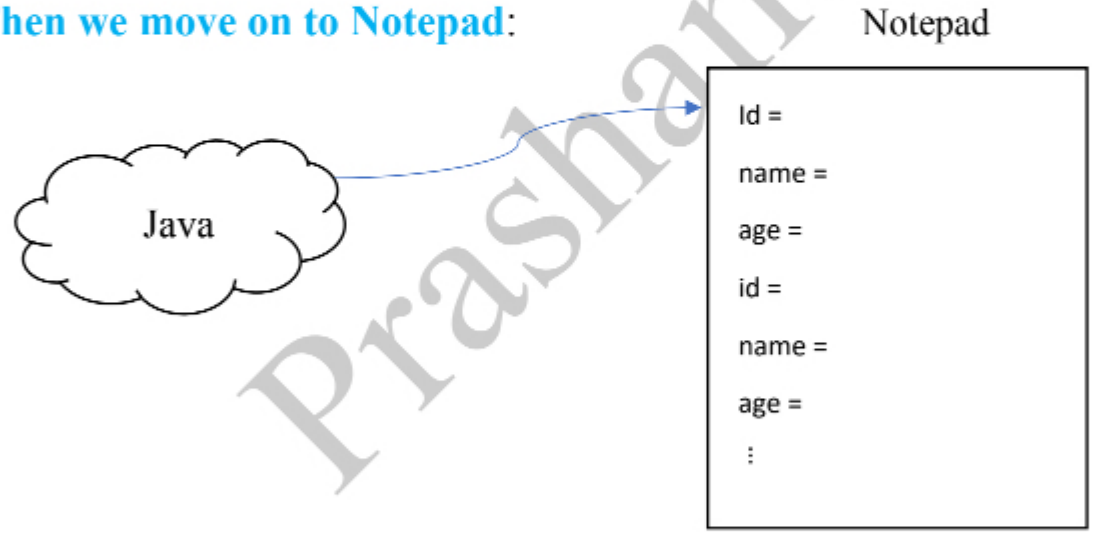
- **Data**: Small information
- Why we need to store a data: In future we retrieve the data and we reuse that data
- What is use of the Storing a data: if we case we store the data in paper after few years we left that paper on that time we lost that data

We already learn core java, in our Java Programming we can store data in variable

- **Variable**: Variable are the Container which is use to store the data.

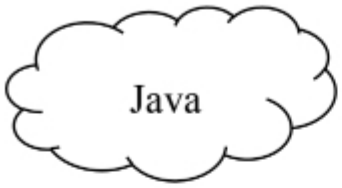
In our Class in java, we can create variable for storing the data and its having disadvantages when we restarting the application that is we can store the data but temporary. But we can store the data permanently.

Then we move on to Notepad:



We can store the data in a Notepad also but, Java compiler it will check line by line the id is present or not, if the id will be present then we retrieve the next 2 lines. If that id is not present it returns null and here handle the data is more difficult.

Then we move on to Excel:



Student

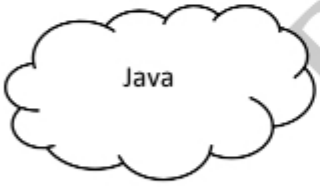
id	name	age	marks

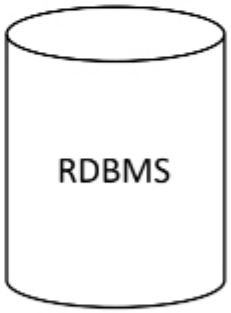
exam test

We can store the data inside a excel sheet, java check the excel sheet row by row and excel sheet increases on that time complexity increases and another problem is creating a relationship between a excel sheet.

Example: Student have details in a Student excel sheet and Student tack a test and information about a test it is present in an exam test excel sheet but we make relationship between Student and exam test excel sheet is very difficult.

Then we move on to DataBase:





We can store the data into a DataBase but data in the form of table but here also have some problems, Database is only known only a Sql language.

- **Sql Language**: it is one of the Query languages which is use to talk with RDBMS

Based E.F. Codd Rules we make differ sql types

Sql language	----	DB Venders
MySQL		Oracle
MSSQL		Microsoft
Oracle		Oracle
DB2		IBM
Sybase		SAP
PlSql		Oracle

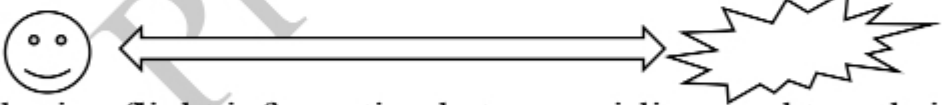
Java can understand only java and DataBase can Understand only SQL but we need DataBase for storing a data.

API:

- Application Programming Interface
- API is a technology
- Which is use to Connect to two application
- JDBC , Rest API , Google Api's , YouTube

Ex:

I. Costumer Server will connect Kitchen and Costumer Kitchen

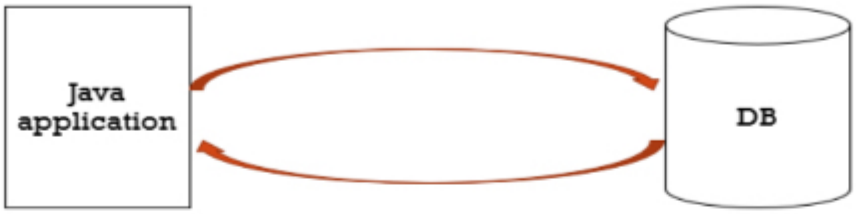


II. Sharing flight information between airlines and travel sites

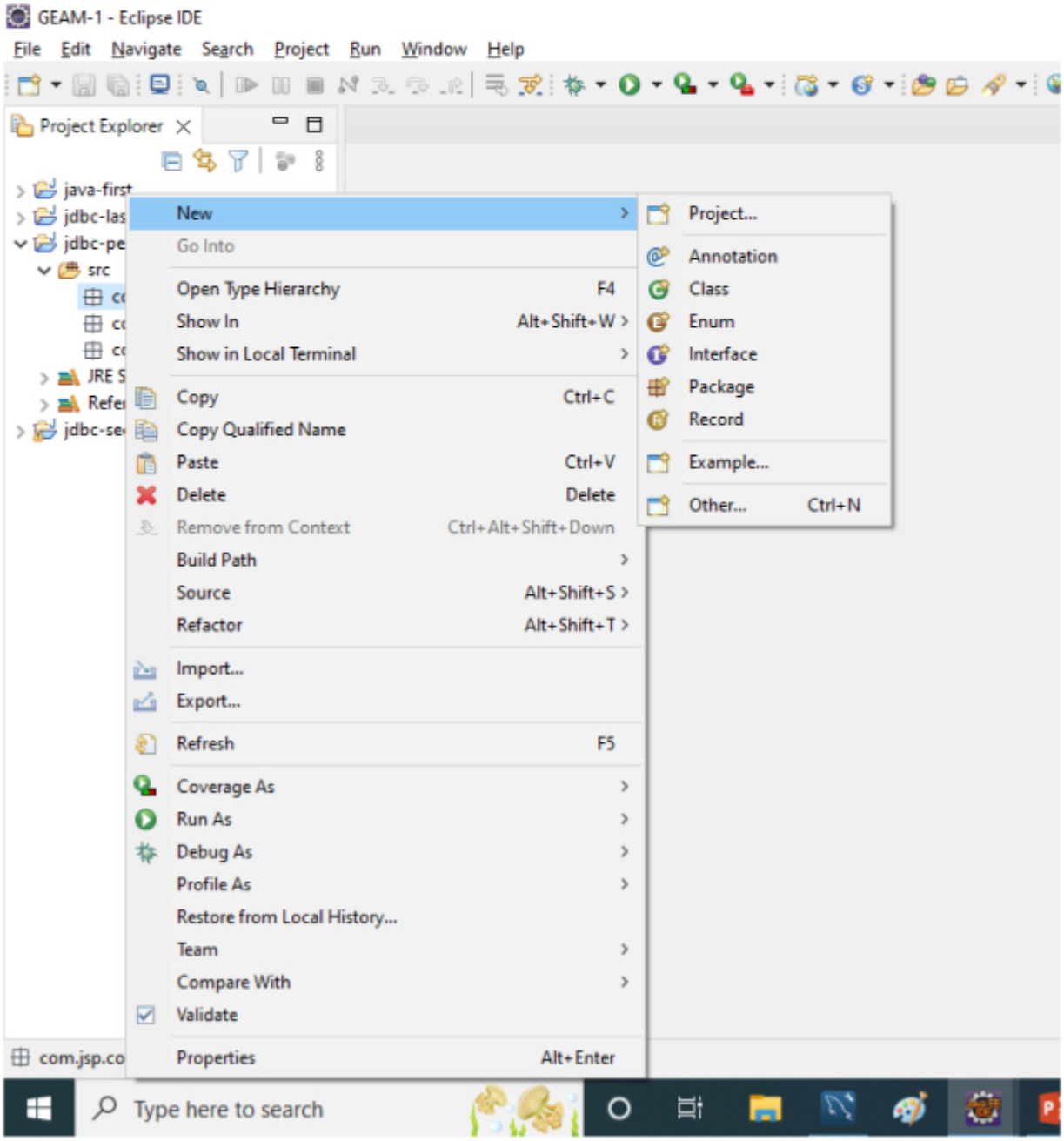
JDBC:

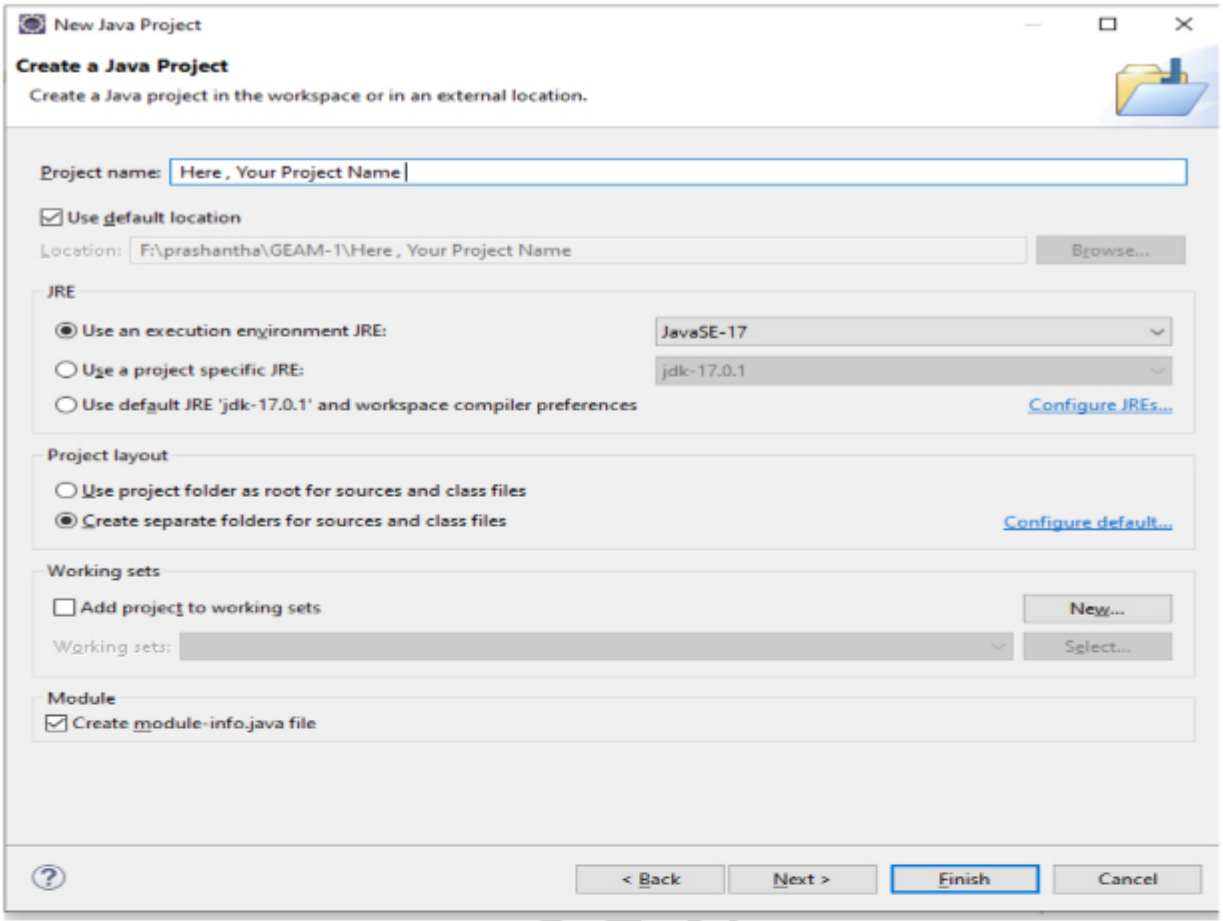
- Java DataBase Connectivity
- JDBC is technology
- JBDC is API
- JDBC is use to establish the connection between java program and database.
- We do not want to install the JDBC explicitly because After JDK 1.0 version, java people purchase the JDBC technology

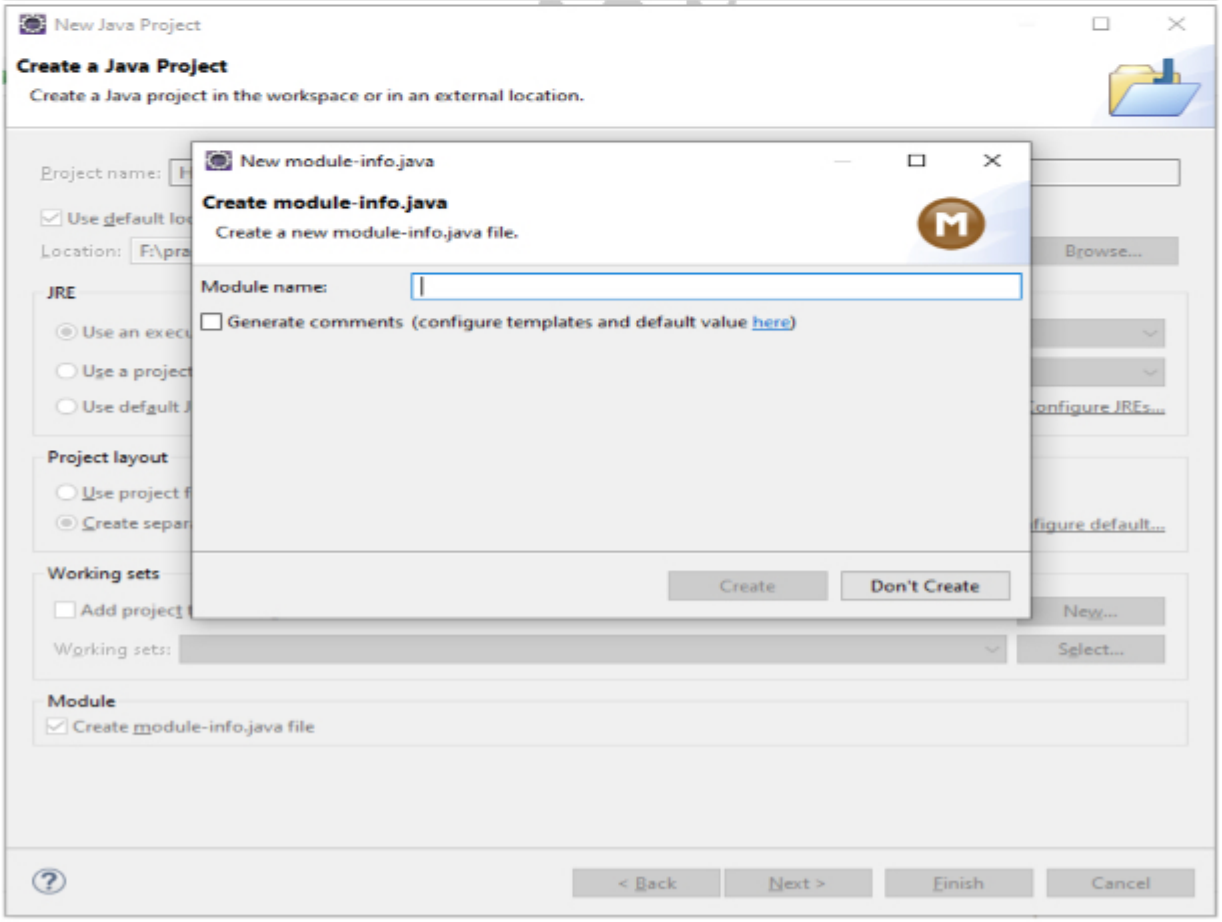
and install into JDBC now it is present inside a jdk only because now days we are using the jdk17 or jdk18 or jdk19.



Creating Normal Java Project :

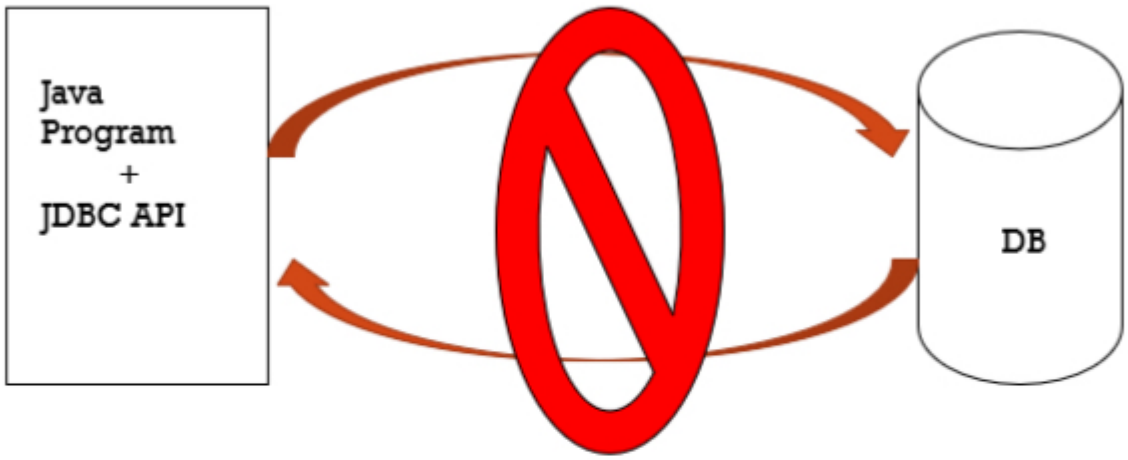



New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: Here , Your Project Name
Use default location
Location: F:\prashantha\GEAM-1\Here , Your Project Name
Browse...
JRE
Use an execution environment JRE:
JavaSE-17
Use a project specific JRE:
jdk-17.0.1
Use default JRE 'jdk-17.0.1' and workspace compiler preferences
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
New...
Working sets:
Select...
Module
Create module-info.java file
< Back
Next >
Finish
Cancel


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
New module-info.java
Create module-info.java
Create a new module-info.java file.
Module name:
Generate comments (configure templates and default value here)
Create
Don't Create
Project name:
Location:
Browse...
JRE
Project layout
Working sets
New...
Working sets:
Select...
Module
Create module-info.java file
< Back
Next >
Finish
Cancel

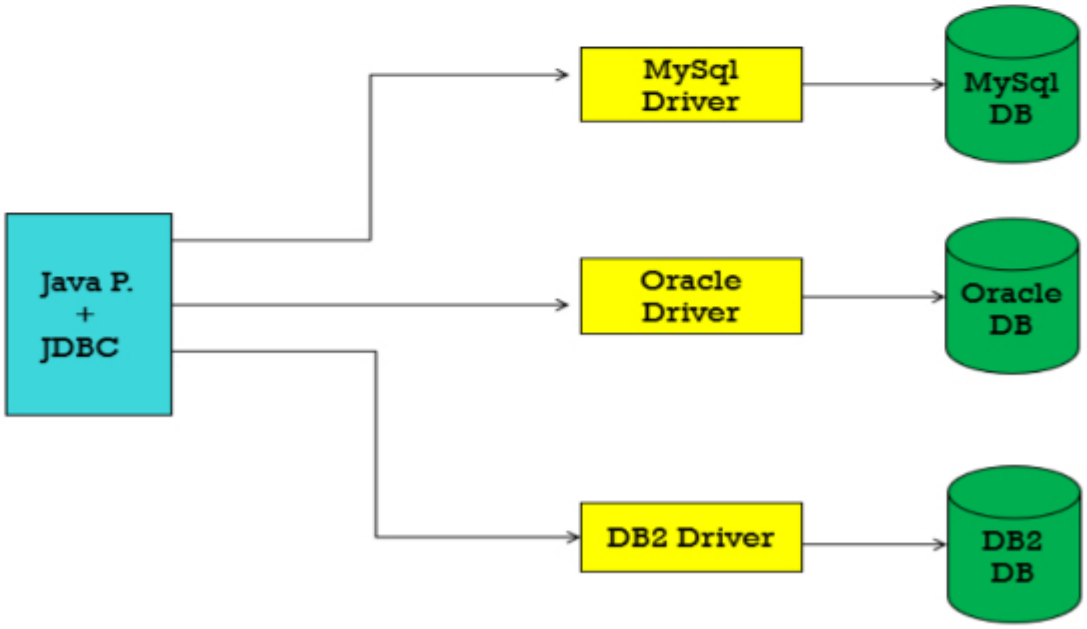
We Follow 5 step to Connect the DataBase with JDBC

- Load/Register Driver.
- Establish the Connection.
- Establish the Statement.
- Execute Statement.
- Close Connection.



- **JDBC can't talk with Database directly**

We are using Driver for connecting the JDBC Api and database



Drivers :

- Driver is software and it is in the form of jar file
- Driver given by respective data base venders

How Can we add the connector into your project:

Download Respective JAR File from Websites

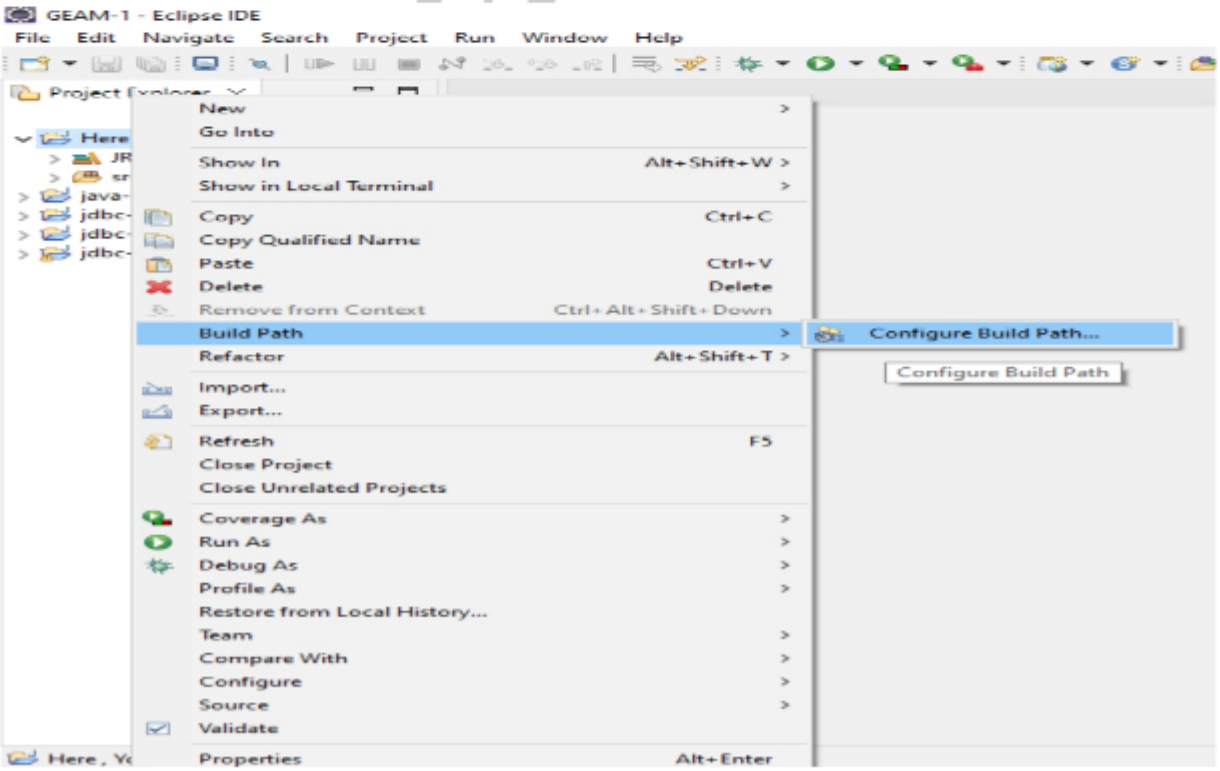
Url link : https://mvnrepository.com or mvnrepository

Search respective RDBMS connectors

Ex :

1. Mysql – search as a **mysql** and then select "**MySQL connector java**" → next select the most usages version and select that version → In that File row click the jar file, it will download the jar file
2. MsSql – search as a **mssql** and then select "**Microsoft JDBC Driver for Sql server**" → next select the most usages version and select that version → In that File row click the jar file, it will download the jar file

Plugin Process :

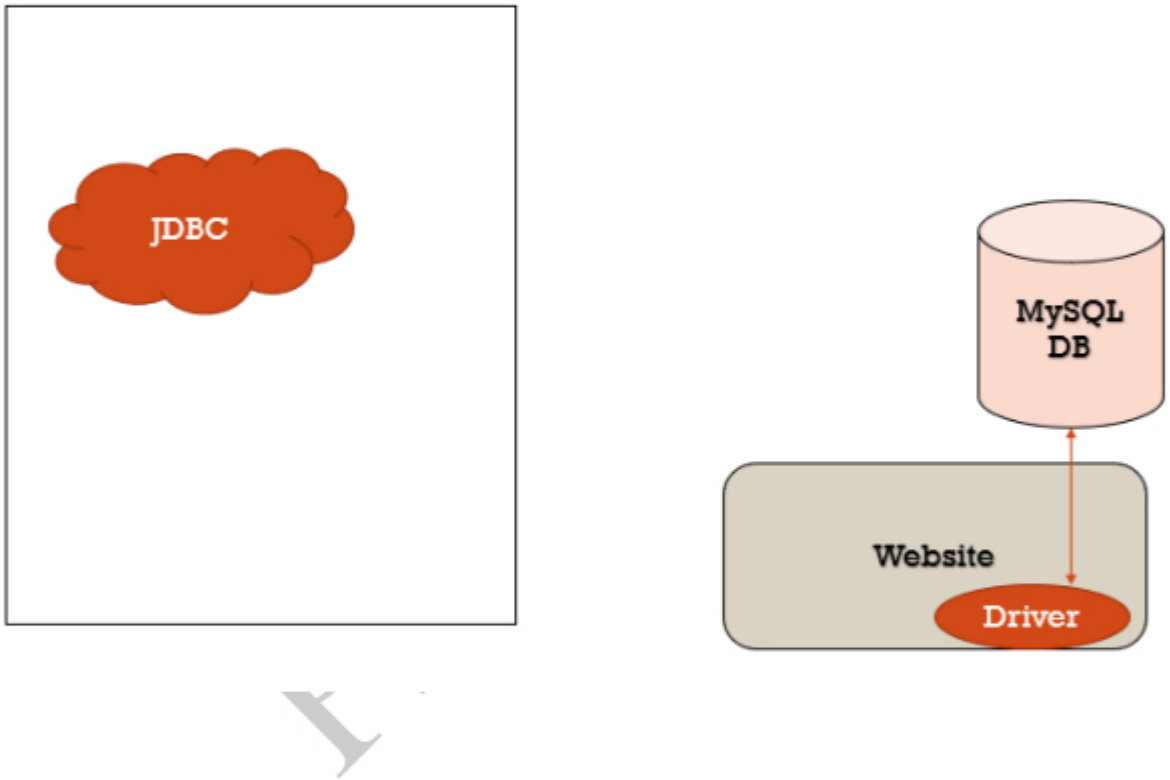


→ Select that **Configure Build Path**

→ Select that **Libraries** and click once on **Modulepath**

→ Select **Add External Jars** button in right side then select your Downloaded jar file and click the open and click that **Apply and close** button .

Before the Load/Register Driver step internal process :



Load/Register Driver :

- With help of this step we can load or register Driver into our Project
- Here , we are loading the driver with in two ways
    1. Load the Driver
    2. Register the Driver

1.Load the Driver:

- It is the first step to connect the database through jdbc.
- It is first way to load the driver

- When we know the fully qualify Driver name on that time only we can use this first way.
- Class.forName("fully qualified class name")
    1. Class→
        a. it is a Class name
        b. it is present inside a java.lang package
    2. forName ("") →
        1. it is a public and static method
        2. it's accepted String parameter
        3. it is present inside a Class class

Ex:
```
package com.jsp;

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

public class FirstStep {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          Connection c = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/ur db name", "root", "ur password");

           System.out.println(c);

        } catch (ClassNotFoundException | SQLException e) {
              e.printStackTrace();
    }
  }
}
```


2.Register the Driver:

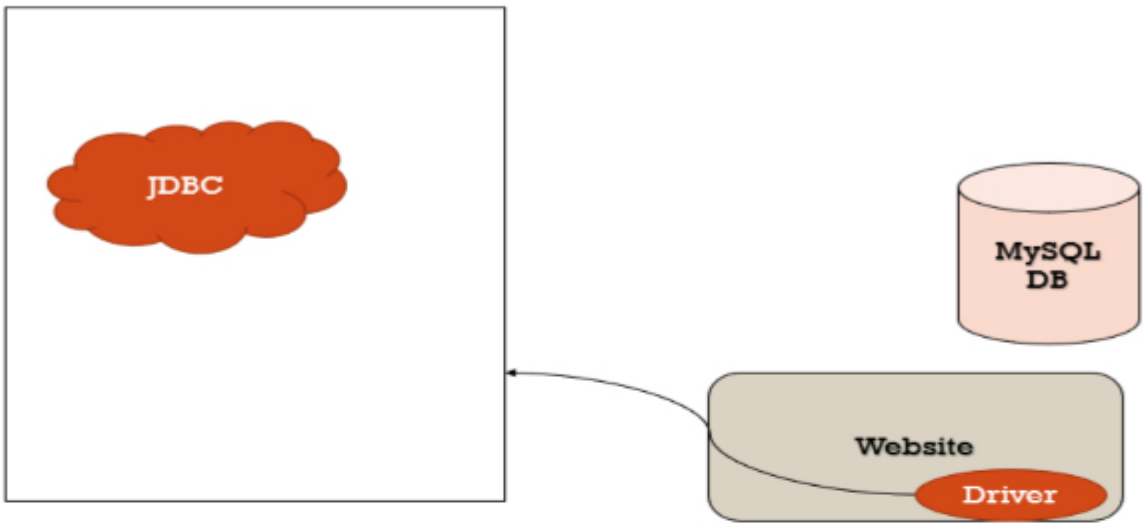
- It is the first step to connect the database through jdbc.
- It is second way to load the driver
- When we don't know the fully qualify Driver name on that time only we can use this second way.

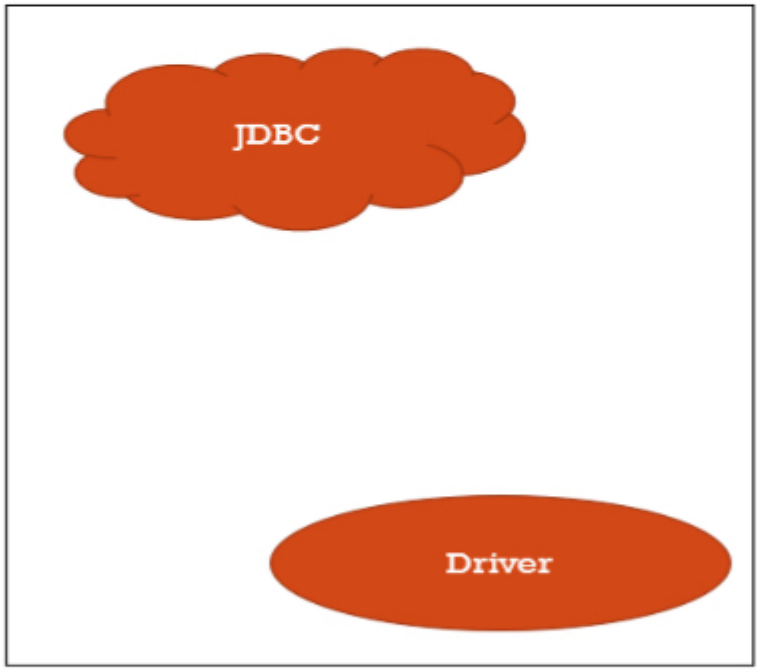
- Compare then load the driver it is bit slower in performance timings.
- Driver d = new Driver();
  DriverManager.registerDriver(d);
    1. Driver →
        a. Driver is a class
        b. Driver is present inside respective database package
    2. DriverManager →
        a. It is a Helper Class
        b. It is present inside a java.sql package
    3. registerDriver ()→
        a. It is public and static method.
        b. It is present inside a DriverManager class
        c. It is accepting object type data.
        d. Inside this method we are passing Driver class object

Ex:

```
package com.jsp;

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

import com.mysql.cj.jdbc.Driver;

public class FirstStep {
	public static void main(String[] args) {
		try {
			Driver driver = new Driver();
			DriverManager.registerDriver(driver);

			Connection c = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name", "root", "ur password");

			System.out.println(c);

		} catch (SQLException e) {
			e.printStackTrace();
		}

	}
}
```

After the Load/Register Driver step internal process :





- **Establish the Connection** :
  - With help of this step we can create the connection in between JDBC and Driver
  - Connection :
    - It is interface
    - It is present inside a java.sql package
    - When we got Connection object then only compiler create connection between jdbc and driver
    - Connection object present in Connection Pool

How can we get Connection object?

1. DriverManager.getConnection(String,String,String) : Connection
2. DriverManager. getConnection(String) : Connection
3. DriverManager. getConnection(String, Properties prop) :Connection

    1. DriverManager →
        a. It is a Helper Class
        b. It is present inside a java.sql package
    2. getConnection()→
        a. it is method
        b. it is present inside DriverManager class
        c. it is a public and static method
        d. its return type is Connection object.

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

- First String Parameter : In this parameter we are passing our url of database
  - Ex : "jdbc: mysql://localhost:3306/Database name"
- Second Parameter : In this parameter we are passing our database name
  - Ex : "root"

- Third Parameter : In this parameter we are passing our database password
  - Ex : “root”

Code :

```
package com.jsp;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class Connection1 {
 public static void main (String[] args) {
        try {
         Class.forName(“com.mysql.cj.jdbc.Driver”);
        Connection c =  DriverManager.getConnection (
        "jdbc:mysql://localhost:3306/your database name","root","root");
        System.out.println(c);
        } catch(ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
        }
 }
}
```


2. DriverManager.getConnection(String) : Connection

- First String Parameter : In this parameter we are passing our url of database , our database user and password
  - Ex : “jdbc:mysql://localhost:3306/database name?user=root&password=root”

- Here, we are passing the url ,user, password in one query.

```
package com.jsp;

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

```
public class Connection1 {
 public static void main (String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");
        Connection c =  DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/your database
        name?user=root&password=root");
        System.out.println(c);
        } catch(ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
        }
 }
}
```

3. DriverManager. getConnection(String, Properties pro) :Connection

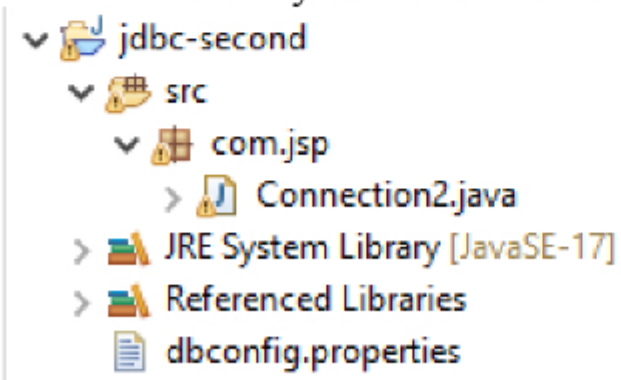
- First String Parameter : In this parameter we are passing our url of database
  - Ex : "jdbc: mysql://localhost:3306/Database name"
- Second Parameter :
  - It is accept Object of Properties class object

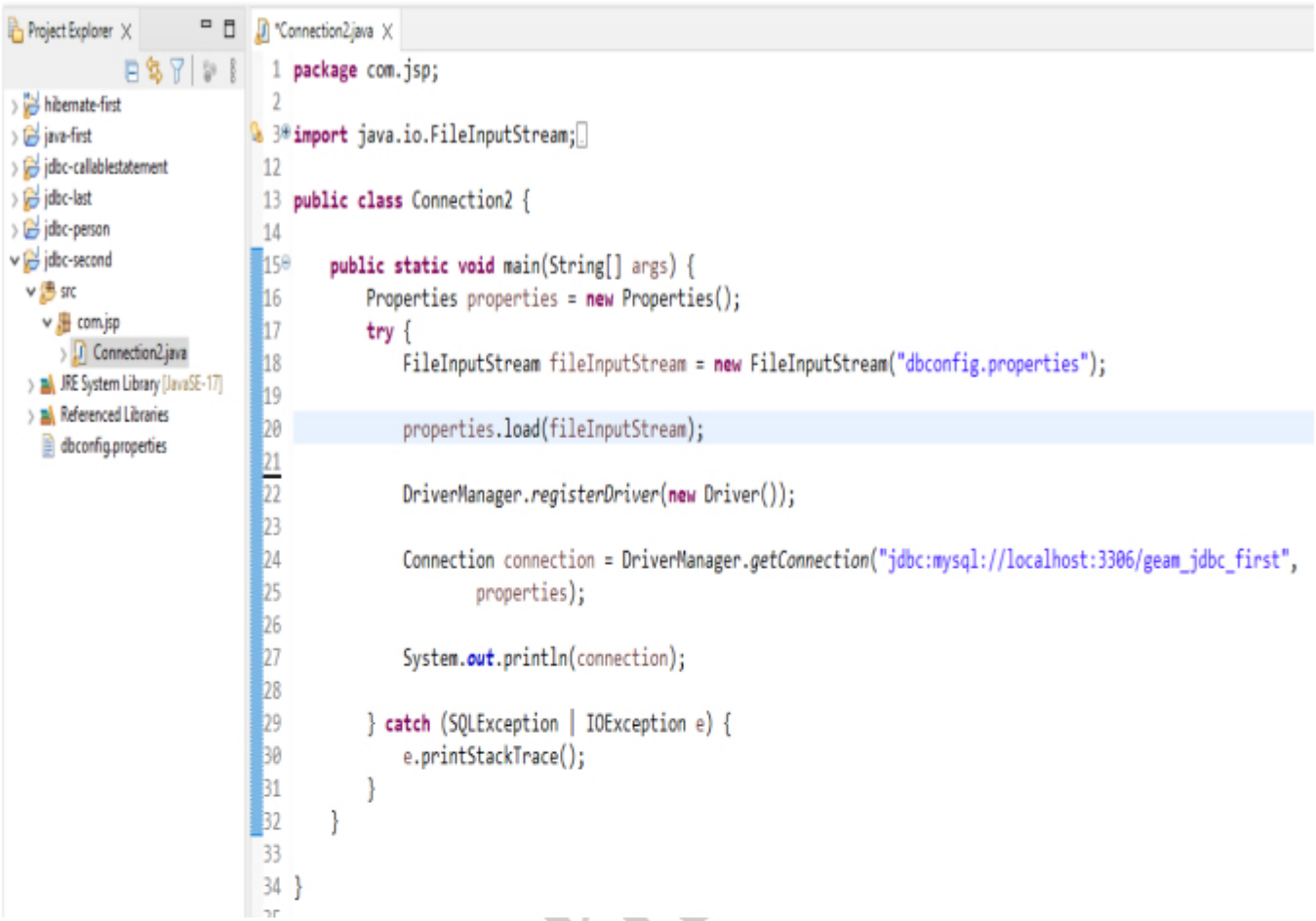
Properties :

1. It is a class and it is present inside a java.sql package
2. Here, we are using some load() method , that method is inside a Properties class

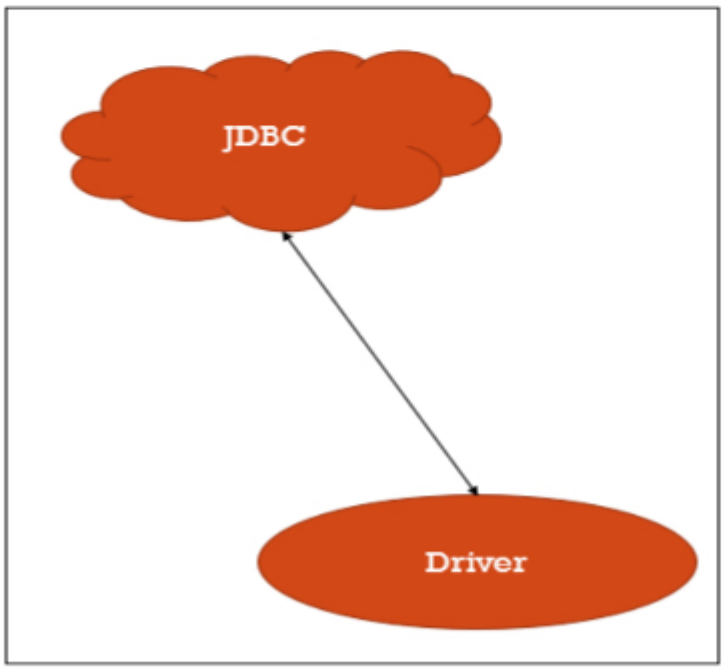
Properties File:

1. It is a File
2. It is Extension with the .properties file
3. It is use to store some configurable data
4. Storing data in the form of key and value pair and based on key name we can fetch the value also.





After the Establish the Connection step internal process :



Establish the Statement :

- With help of this step we can transfer the sql query form ur java into database.
- Statement → PreparedStatement → CallableStatement
- **Statement** :
  - It is interface and it is present inside a java.sql package
  - Here , we are passing static query. (not mandatory)
  - With help of the statement object we can transfer the sql query form ur java into database.
  - Object Creation :
    - connection.createStatement( ) ;

    connection → it is reference of the Connection obj

    createStatement() → it is method ,it is public and non static method , it is present inside a Connection interface.

Ex:

```
package com.jsp;

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

public class FirstStep {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

        Connection c = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name","root","ur  password");

          Statement statement = c.createStatement();

          System.out.println(statement);

        } catch (ClassNotFoundException | SQLException e) {
```

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

- **PreparedStatement** :
  - It is interface and it is present inside a java.sql package
  - Here , we are passing dynamic query. (not mandatory)
  - With help of the statement object we can transfer the sql query form ur java into database.
  - It is a child of Statement interface.
  - Object Creation :

    connection.prepareStatement(“Sql Query”) ;

    connection → it is reference of the Connection obj

    prepareStatement(“ sql query”) →

    - It is method.
    - It is public and non-static method
    - It is present inside a Connection interface
    - It is accepting string parameter and inside that parameter we are passing our sql commend

Ex :

```
package com.jsp;

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

public class TaskC {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          Connection connection = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name", "root","ur password");
```

```
            PreparedStatement preparedStatement = connection.prepareStatement("insert into
table_name values(?,?,?)");

            preparedStatement.setInt(1, 1);
            preparedStatement.setString(2, "Radhe");
            preparedStatement.setInt(3, 24);

            System.out.println(preparedStatement);

        } catch (ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
        }
    }
}
```

Statement	Prepared Statement
It is Base interface	It is Extends from Statement interface
It is used to write a Static Query.	It is used to write a Dynamic query because it's having a Placeholder.
It is used for DDL statement	It is used for any SQL Query
It is load every time when we execute.	It is load once when we execute and print any time.
It's performance is slower than Prepared statement because when we call execute every time class loaded	It's performance is faster than Statement because It load only once and print any time .

- **CallableStatement** :
  - It is also one of the ways to transfer the sql commend from java into database
  - It is also interface.
  - It is child of PreparedStatement interface.
  - It is faster than PreparedStatement.
  - But here , we are using Stored Procedure

- Object Creation :

connection.prepareCall(“call procedure_name(?,?,?) ”);

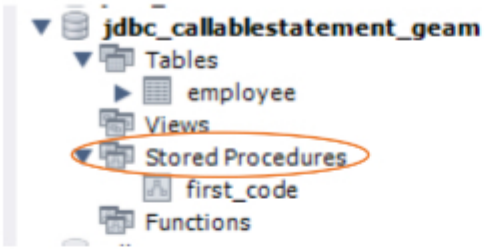
connection → it is reference of the Connection obj

prepareCall(“call procedure_name(?,?,?) ”); →

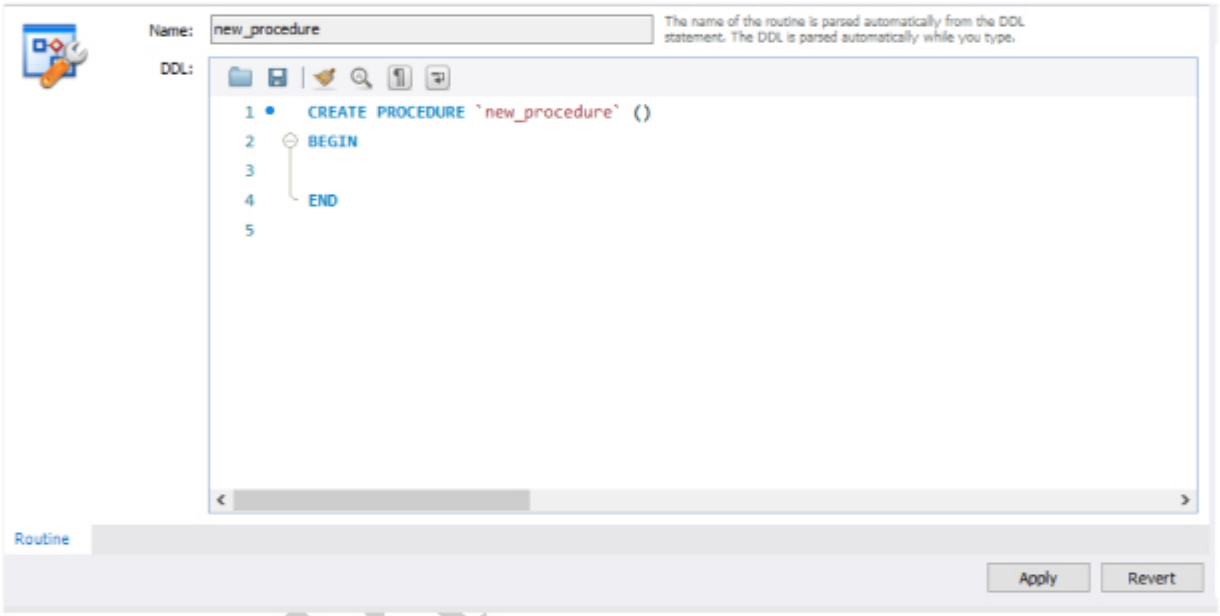
- It is method.
- It is public and non-static method
- It is present inside a Connection interface
- Inside the method we are calling stored procedure with help of the call

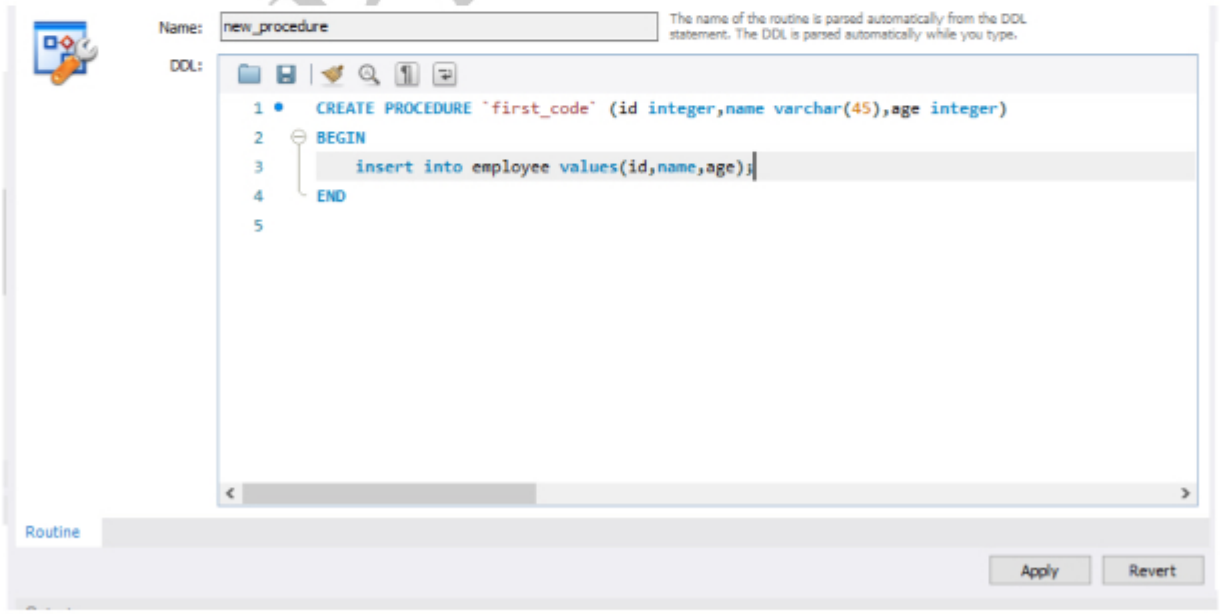
Ex :

```
package com.jsp;

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

public class TestEmployee {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          Connection connection = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/jdbc_callablestatement_geam?user=root&password=Prashi@123");

            CallableStatement callableStatement = connection.prepareCall("call
        first_code(?,?,?)");

          callableStatement.setInt(1, 2);
          callableStatement.setString(2, "Paru");
          callableStatement.setDouble(3, 10000);

          System.out.println(callableStatement);

         } catch (ClassNotFoundException | SQLException e) {
                  e.printStackTrace();
          }
   }
}
```



- Right click on stored procedure





- By last Apply button.

Execute the Statement :

- With help of this method we can execute the sql commends.
- Here, in this step we are using mainly 3 methods
  - execute("string sql") : boolean
  - executeUpdate("string sql") : int
  - executeQuery("string sql") : ResultSet

execute("sql")	executeUpdate("sql")	executeQuery("sql")
The method can be used for any kind of sql statement	The method can be used for update or modify the database	The method can be Used for which Sql statement retrieves some data from the database
The method returns boolean	The method returns int	The method returns ResultSet
The method is used to both execute select and non selected queries.	The method is used to execute only non select queries.	The method is used to execute only select queries.
Ex : insert,select,update... (any type of sql queries)	Ex : insert ,update, delete, create ,alter…..	Ex : select

Note :

1. If we are using Statement in third step, in this step only we are passing our sql commends.
2. If we are using PreparedStatement in third step, in this step we are not passing our sql commends.

Ex :

1. Here , we are using execute( )

```
package com.jsp;

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

public class TaskC {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          Connection connection = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name", "root","ur password");

          PreparedStatement preparedStatement = connection.prepareStatement("insert into
table_name values(?,?,?)");

          preparedStatement.setInt(1, 1);
          preparedStatement.setString(2, "Dimple");
          preparedStatement.setInt(3, 24);

          preparedStatement.execute();

          System.out.println("Data Saved");

        } catch (ClassNotFoundException | SQLException e) {
                    e.printStackTrace();
          }
   }
}
```

2. Here , we are using executeUpdate()

```
package com.jsp;

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

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

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

          Connection connection = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name", "root","ur password");

          PreparedStatement preparedStatement = connection.prepareStatement("insert into
table_name values(?,?,?)");

          preparedStatement.setInt(1, 1);
          preparedStatement.setString(2, "Dimple");
          preparedStatement.setInt(3, 24);

          preparedStatement.executeUpdate();

          System.out.println("Data Saved");

        } catch (ClassNotFoundException | SQLException e) {
                  e.printStackTrace();
         }
   }
}
```

3. Here , we are using executeQuery()

```
package com.jsp;

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

public class TaskC {
  public static void main(String[] args) {
        try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          Connection connection = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/ur db name", "root","ur password");

          Statement statement = connection.createStatement();

          ResultSet resultSet = statement.executeQuery("select * from table_name");

          System.out.println(resultSet);

        } catch (ClassNotFoundException | SQLException e) {
                  e.printStackTrace();
         }
```

```
    }
}
```

- **ResultSet** :
  - It is Interface and it is present inside a java.sql package
  - It is acts like a pointer
  - When we get a ResultSet object immediately resultset create one temporary table then it will store all the data into in that table.
  - How can we fetch the data from ResultSet means here we are using while block

```
while(resultSet.next())
{
    Product product = new Product();
    product.setId(resultSet.getInt(1));
    product.setName(resultSet.getString(2));
    product.setBrand(resultSet.getString(3));
    list. add(product);
    System.out.println(product);
}
```


next( ) : with help of the next( ) that pointer is move to next row in ResultSet

getInt( ) ,getString( ) ,getDouble( ) etc.. : with help of this we can fetch the data from ResultSet

Batch Execution :

- Instead of executing a single query, we can execute a batch (group) of queries
- It makes the performance fast.

- We can execute group or set of query in one shot.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing .

```
PreparedStatement preparedStatement =
                        connection.prepareStatement("str");
```

Ex :

```
for(Employee e : arraylist){
    preparedStatement.setInt(1,e.getId());
    preparedStatement.setString(2,e.getName());
    preparedStatement.setString(3,e.getEmail());
    preparedStatement.addBatch();
}
preparedStatement.executeBatch();
System.out.println("All the Data Added");
```


Close the connection

- When you obtained a connection object with DriverManager.getConnection() successfully, you can create statement objects and execute them with this connection object. But when you are done with this connection, you should close it with help of the close() .
- How to Close the DataBase Connection :
  - connection.close();

Why we need to close the connection with database ?

- For the purpose of safe coding, you should always close database connections explicitly to make sure that the code was able to close itself gracefully and to prevent any other objects from reusing the same connection after you are done with it.