



Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
	Kevin	Edward	2440081614

Course Code	: COMP-6699	Course Name	: Object Oriented Programming
-------------	-------------	-------------	-------------------------------

Class	: L2BC	Name of Lecturer(s):	Jude Joseph Lamug Martinez, MCS
-------	--------	----------------------	---------------------------------

Major	: Computer Science
-------	--------------------

Title of Assignment: Employee Database Manager Application
(if any)

Type of Assignment : Final Project

Submission Pattern

Due Date :	22/06/2021	Submission Date	: 19/06/2021
------------	------------	-----------------	--------------

The assignment should meet the below requirements.

Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.

Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.

The above information is complete and legible.

Compiled pages are firmly stapled.

Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student :



Edward Kevin

Contents

Project Specification	Pg. 4
Solution Design	Pg. 6
Notes	Pg. 12
Implementing GUI & Database Connection	Pg. 13
Structure	Pg. 16
References	Pg. 17

Project Specification

The purpose of creation of this project is to provide a media for all its users to interact with / manipulate a pre-defined SQL Table which functions by listing all details of each virtual employee which has been recorded in the RDBMS server(MySQL) through a graphical user interface application developed purely in Java using its own built-in gui design tool, most commonly referred to as "Swing" and an external API known as MySQL Connector/Java to establish a form of connection to the SQL table(JDBC).

The application itself currently supports three main features where they allow insertions(INSERT INTO), updates(UPDATE SET *ARGS WHERE ID = ?) and deletion(DELETE FROM) of a specific employee record which has either existed or not in the table, these operations are all meant to be done from within the displayed GUI, under the condition that each user has already defined and run their own employee tables containing fixed names of its schema(SQL Package) and the name of the table itself, which have been described in the database file in the contents of this repository.

Netbeans(Version 12.0) is the IDE that was used during the development of this project. The reason for this is that it comes with a feature which allows every Java project made within its global folder to gain a direct access into its built-in global library which offers quite a lot of the most commonly used frameworks and APIs by many developers in the world while working with Java. They will no longer need to manually include JAR files of a specific framework into their project's classpath which consumes time and a lot of individual effort. The mentioned global library also includes the MySQL Connector/Java API which served as the backend database server needed for the application to work properly.

Below is a list of both internal and external packages, as well as a vital library used in the development of this project :

1. **Swing**(Internal | IDE-Dependent)
A built-in framework which serves as a GUI widget toolkit for Java. Built on top of the AWT API and written entirely in Java as well. A specific feature however, a direct JFrame design tool available in both Eclipse and Netbeans IDEs require an external swing .jar file which provides the package "**org.lib.awtextra**" so that its main java class is not being omitted anymore.
2. **AWT(Abstract Window Toolkit)**(Internal)
This built-in API is used to decorate and provide styles to every GUI component in the application. However, the primary objective of using AWT is to provide Java interfaces that functions by taking

and returning a series of responses(**ActionListener**↔**ActionEvent**, **ItemListener**↔**ItemEvent**) from specific components that allow interaction with the user(**JButton**, **JComboBox**).

3. **java.sql Package**(IDE-Dependent)

The interfaces available in this package are built-in and functions by executing a specific static block in the class "Driver" which initiates a connection to an user's SQL table running in their private home IP-address(Localhost | 127.0.0.1). This package requires the external API "MySQL Connector/Java" in order to work properly.

4. **MySQL Connector/Java**(External)

Since NetBeans IDE has been updated to version 12.0 , its global library does not support a direct classpath import for this API anymore. Ever since then, every available Java IDEs require a manual JAR file import to a project's classpath to register this API so that the java source file can access it. While attempting to load this into a java file, be advised that its developers have **deprecated** the old driver class "**com.mysql.jdbc.Driver**" and made a new one under the package name "**com.mysql.cj.jdbc.Driver**". The "**Driver**" class there is the one that has been mentioned to be containing a specific static block, of which if it were to be called using the method "**Class.forName(String pkg)**", then the java source file will make an attempt to connect to a database server using the specified Username, Password And Port Number from which the user has entered during the set up of their own MySQL servers. If the attempt was unsuccessful then an external "**SQLException**" is thrown from which it could be handled to instead return a message that notifies the user rather than returning a raw error object by using a try and catch statement.

Solution Design

The application developed from this project requires a constant access to a running MySQL database server in every user's local environment(**Start→Services**). That of course combined with the appropriate server details along with the syntax of the SQL Table which is present in the GitHub repository tied for this project.

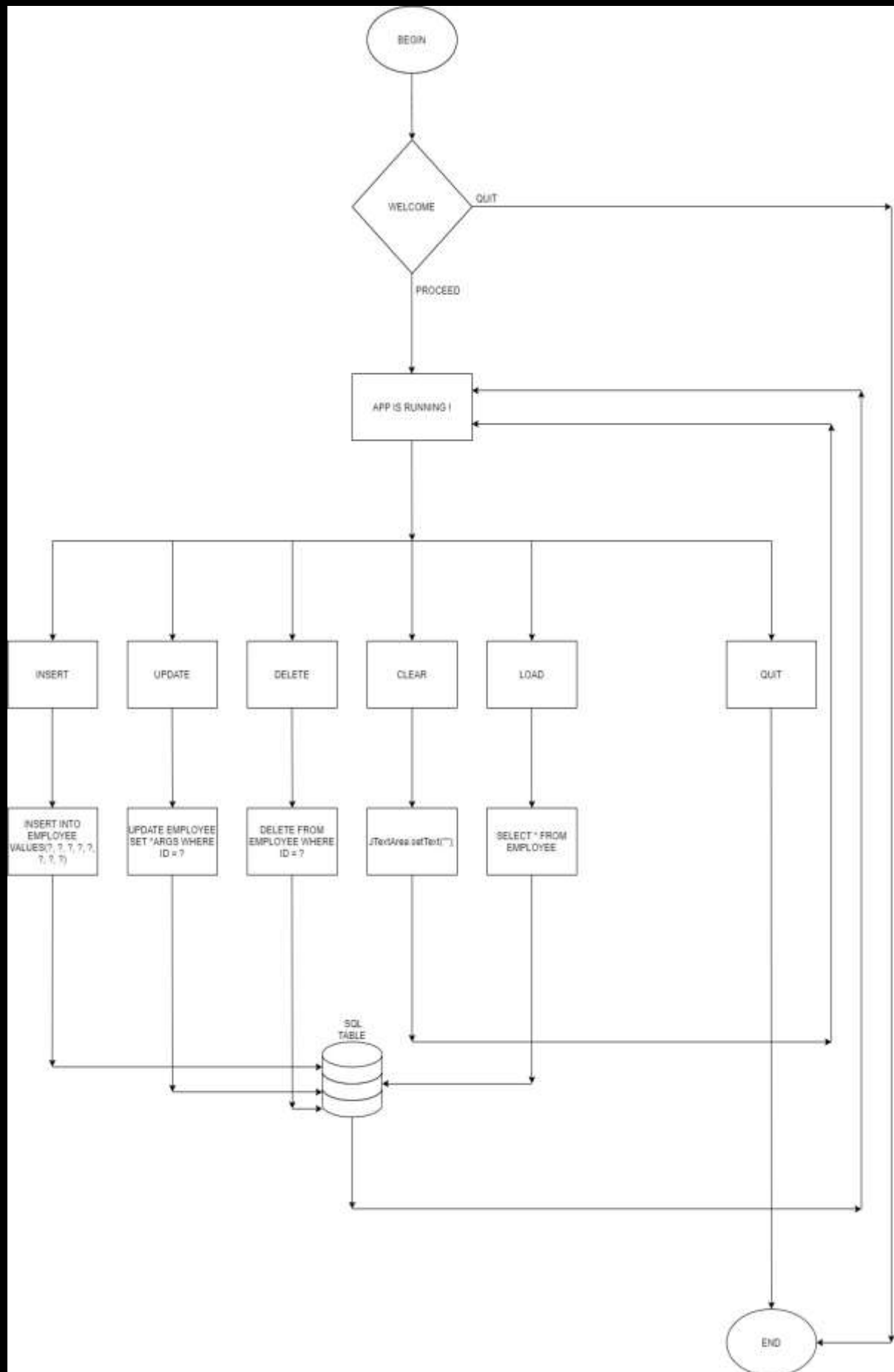


If an user would like to know how to set up her / his own MySQL server to run this project, she / he has been provided a link to an installer from the repository which directs to the official documentation of MySQL providing a downloadable folder, of which is the installer itself. The next steps will be provided by the prompt to guide them into finally acquiring an running an active database server in each of their own environments.

Once the database server is running, the user will need to open either the terminal or a workbench related to the active instance of the server, and create the SQL Table by following the syntax listed in SQL file in the GitHub repository.

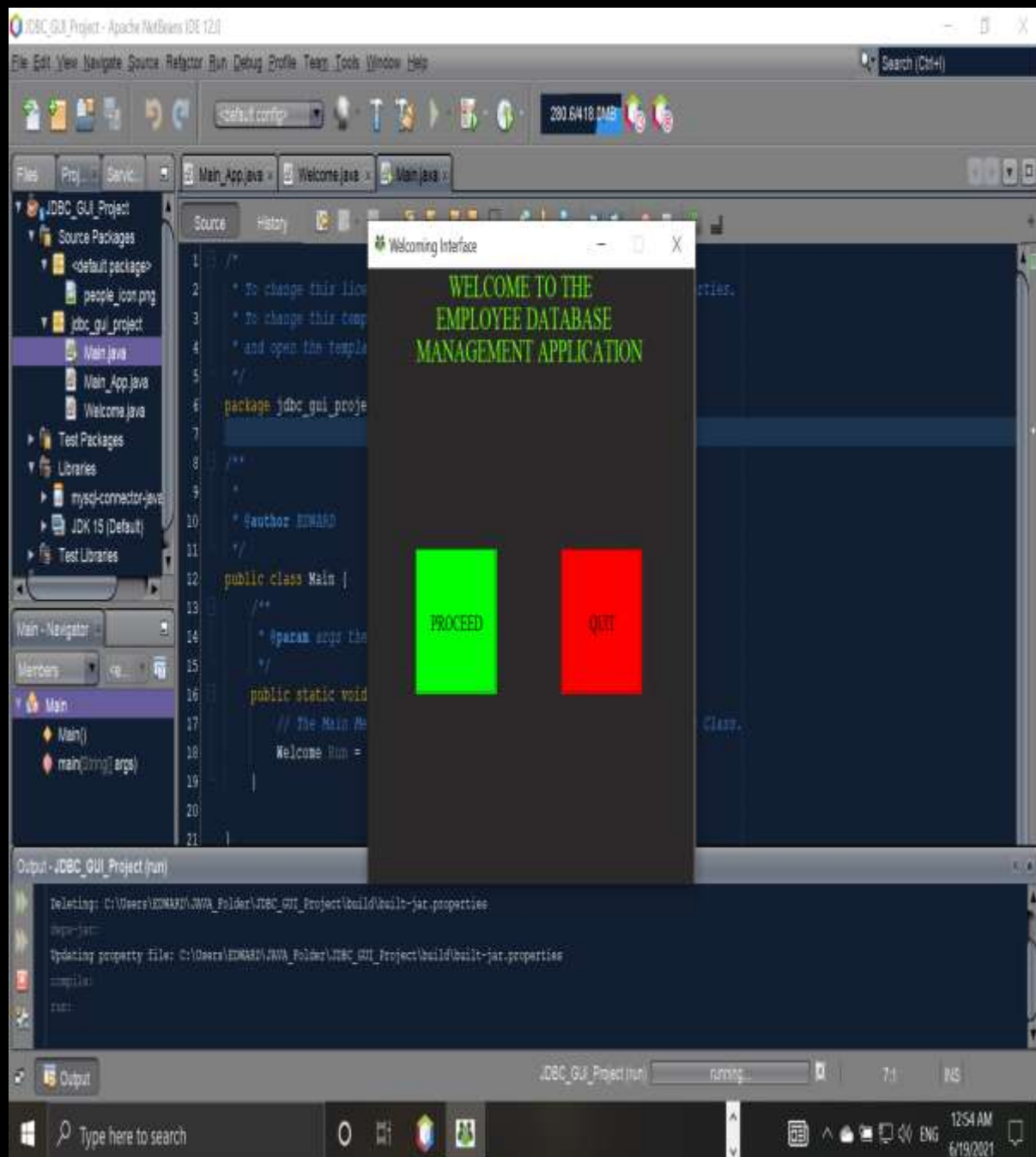
If the table has been successfully made and is giving proper responses to queries, then they are now ready to finally run the application. All they have to do is to activate a GitBash terminal, clone the repository, navigate into the java package, compile and run the main class and then the application will run. Everytime the user interacts with the app, their SQL tables will be affected as well since the app is connected to it.

In order to illustrate the working of the application better, below is a flowchart visualization of the structure of the app :

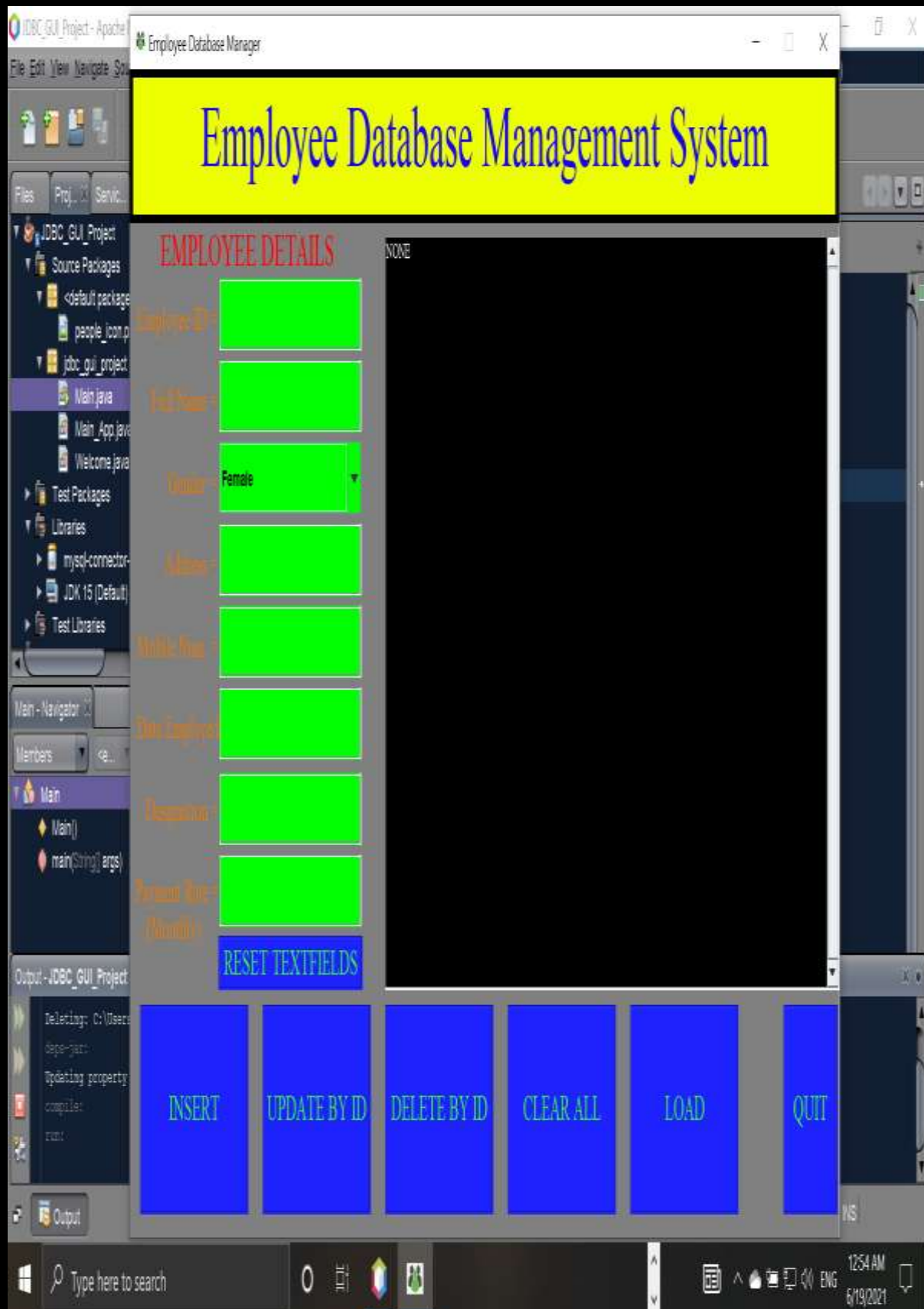


The flowchart of the structure of the application above has clearly described that the existence of the SQL table containing details of virtual employees is of the utmost importance for the application to work properly, as its main duty is to manipulate the table according to the user's needs and wants, nothing more or less, plain and direct.

The current user interface(display) of the application is shown below :



The Welcoming UI



The Main Application With Empty JTextArea



The JTextArea Has Been Filled With Two Records From SQL Table

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator Employee_Table x

Limit to 50000 rows

SQL Additions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

SCHEMAS

Filter objects

company

Tables

employee

Views

Stored Procedures

Functions

sakila

sys

world

1 SHOW DATABASES;

2

3 USE COMPANY;

4

5 /*

6 THIS SUPPRESSES QUERY WARNINGS.

7 */

8 SET SQL_SAFE_UPDATES = 0;

9

10 SELECT * FROM EMPLOYEE;

Result Grid

Filter Rows: Export: Wrap Cell Content: Result Grid

Employee_ID	Name	Gender	Address	Mobile	Date_Employed	Designation	Monthly_Pay
431083	Daisy Duck	Female	The MickeyMouse Clubhouse	+19438103811	18/01/2021	Sr. Database Administrator	US\$ 13,08
120429	Donald Duck	Male	The MickeyMouse Clubhouse	+19361042851	17/01/2021	Sr. Data Scientist	US\$ 12,93

Administration Schemas

Information

No object selected

EMPLOYEE 3 x

Read Only Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
3	00:53:33	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.000 sec
4	00:53:36	SELECT * FROM EMPLOYEE LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
5	00:58:28	SELECT * FROM EMPLOYEE LIMIT 0, 50000	2 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Type here to search

12:58 AM 6/19/2021

The Java Application Affects The Backend RDBMS(MySQL)

Extra Notes

- 1) For users who may need a guide on installing and initializing their own MySQL servers, along with the steps to create a new SQL table, a guide has been posted under the section "**Extras**" in the GitHub Repository.
- 2) Double check if the cloned java application has included the external "MySQL Connector/Java" API in the project's classpath.
- 3) During the creation of the SQL table, please kindly follow the provided syntax in the GitHub Repository. Any mistakes / mismatch occurred during the creation of the table will not be tolerated by the Java application, which results in either Java warnings / SQL exceptions being thrown.
- 4) If The User's Specified MySQL Server Details Were Any Different Compared To The Picture Below During Installation, Please Manually Change The Value Of These String Attributes So That The Application Could Run :

```
// MySQL Server's Details.  
  
private final String URL = "jdbc:mysql://localhost:3306/COMPANY";  
private final String UserName = "root";  
private final String Password = "12345";
```

- 5) For users who were using any other IDEs in comparison to NetBeans, kindly install the swing layout JAR file dependency from the GitHub Repository and double check if it's been added to the project's classpath, just to ensure that the project has full access to all features of the swing framework, including the JFrame design window. If the swing dependency isn't there yet, manually include its JAR file in the classpath.
- 6) For convenience purposes, **please try not to use Visual Studio Code Editor to run this project** as there may be unknown difficulties and warnings coming if they do occur. Use an IDE that's been specifically and primarily made to run and manage Java programs and packages.

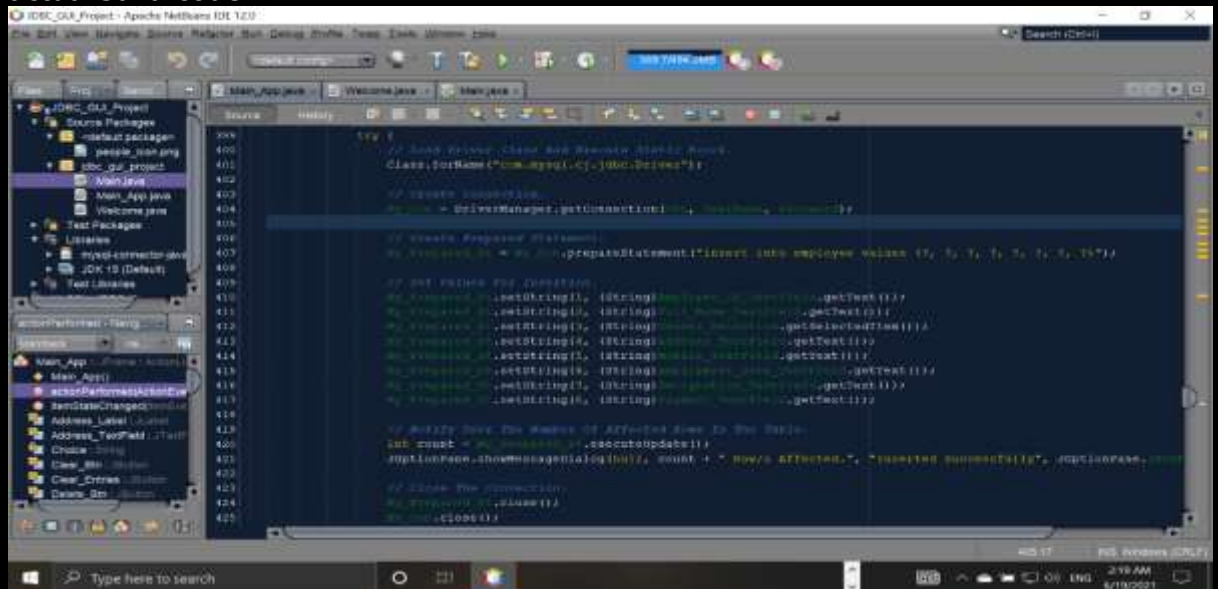
Project Implementation Techniques

The implementation of the application was pretty straightforward. All that needs to be done were to ensure that each JButton properly establishes a connection to SQL Table and executes a specific type of query which either fetches / manipulates the contents of the table. According to Navin Reddy(2016, March 30), from his online Youtube tutorial regarding Java Database Connectivity Tutorial, there are seven steps that has to be taken into account to successfully connect a Java source file into an RDBMS :

- 1) Import the package for JDBC(**java.sql.***).
- 2) Register & Load The Driver Class Each Time A Query Needs To Be Executed.
- 3) Establish Connection To RDBMS Using **java.sql.Connection** Interface.
- 4) Creating A Statement(**java.sql.Statement**) Interface Specifically For DQL Queries. For DML Queries, A Different Kind Of Statement Has To Be Used, Of Which Is The PreparedStatement(**java.sql.PreparedStatement**) Interface.
- 5) Execute The Specified SQL Query, And Store Its Return Value In An Object Instance Of The ResultSet Interface (**java.sql.ResultSet**) Interface.
- 6) Process The Obtained Results By The User's Self Preference. User'll Have To Decide On Their Own About What To Do With The Obtained Table Records.
- 7) Make Sure That Statement And Connection Interfaces Are Closed When The User Has Finished Using Them. To Do This, Execute **java.sql.Statement.close();** Or **java.sql.PreparedStatement.close();** For Prepared Statements, And Finally **java.sql.Connection.close();** .

The development process of the project stuck strictly with the seven steps mentioned above and as a result in the end, the application was successfully communicating with the SQL Table and also was able to remotely fetch and manipulate its initial contents. Inserting new records into the table has worked fine as well. Below is how the seven steps were implemented in

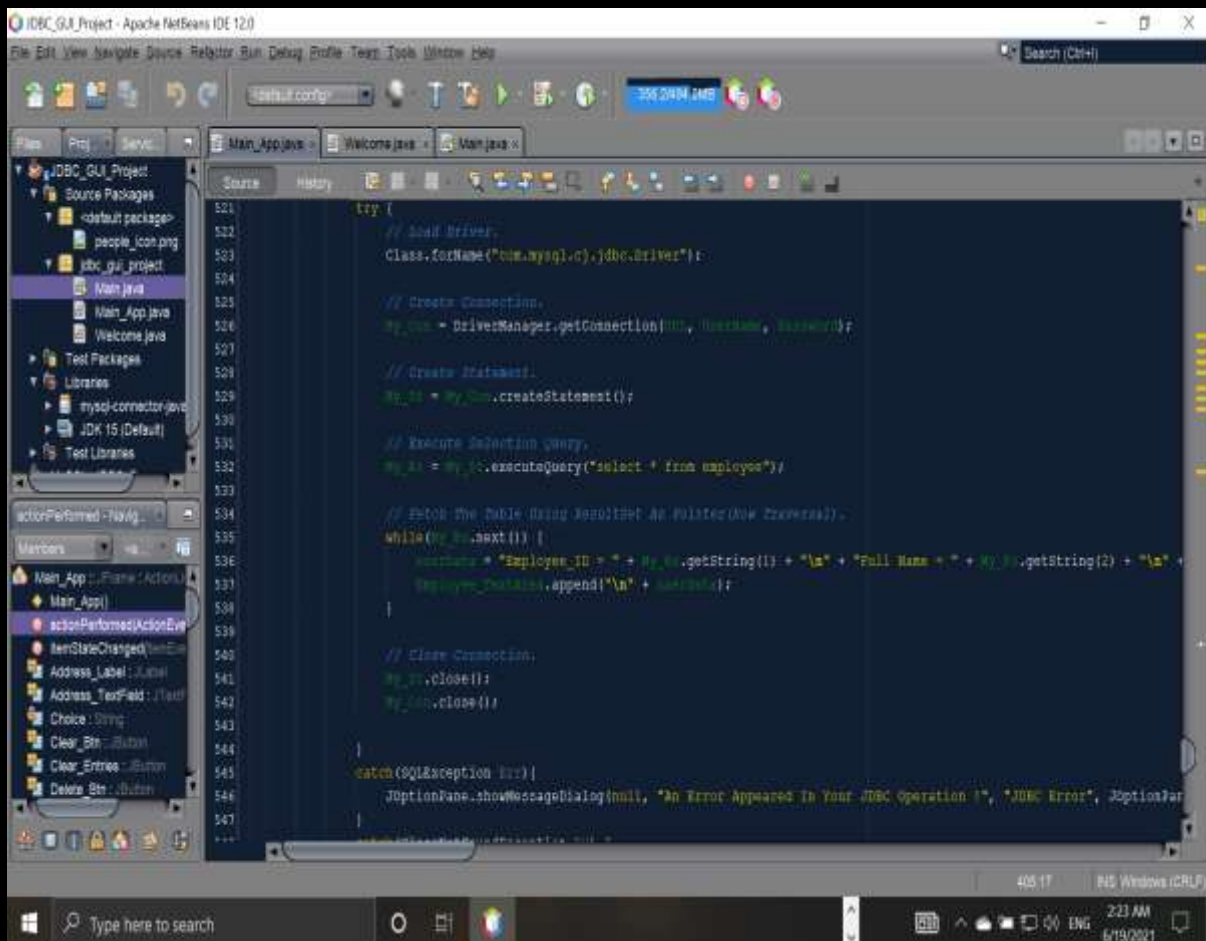
actual Java code.



The screenshot shows the Apache NetBeans IDE with a project named 'JDBC_GUI_Project'. The 'Main.java' file is open, displaying Java code for a DML Insert Query. The code includes comments and comments for creating a connection, creating a statement, and executing an insert query. The code is as follows:

```
try {  
    // Load Driver. Class and Register Driver.  
    Class.forName("com.mysql.cj.jdbc.Driver");  
  
    // Create Connection.  
    MyConn = DriverManager.getConnection(url, username, password);  
  
    // Create Prepared Statement.  
    MyStatement = MyConn.prepareStatement("insert into employee values (?, ?, ?, ?, ?, ?)");  
  
    // Get values for insertion.  
    MyEmpID = Integer.parseInt(jTextField1.getText());  
    MyEmpName = jTextField2.getText();  
    MyEmpSalary = Integer.parseInt(jTextField3.getText());  
    MyEmpAddress = jTextField4.getText();  
    MyEmpCity = jTextField5.getText();  
    MyEmpState = jTextField6.getText();  
    MyEmpZip = Integer.parseInt(jTextField7.getText());  
  
    // Execute the query.  
    MyStatement.setInt(1, MyEmpID);  
    MyStatement.setString(2, MyEmpName);  
    MyStatement.setInt(3, MyEmpSalary);  
    MyStatement.setString(4, MyEmpAddress);  
    MyStatement.setString(5, MyEmpCity);  
    MyStatement.setString(6, MyEmpState);  
    MyStatement.setInt(7, MyEmpZip);  
  
    // Execute the query.  
    MyStatement.executeUpdate();  
  
    // Close the connection.  
    MyStatement.close();  
    MyConn.close();  
}
```

Performing A DML Insert Query

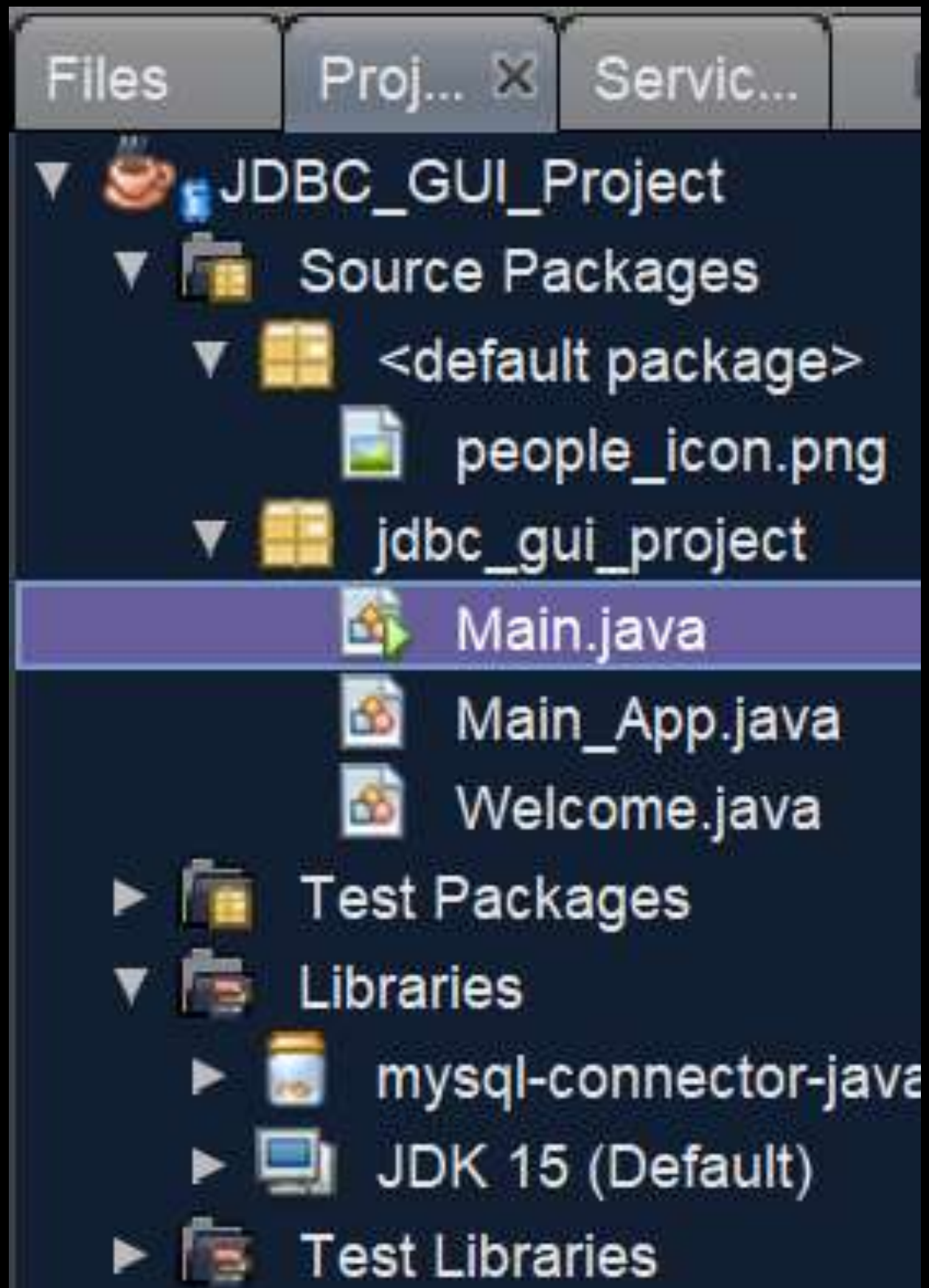


The screenshot shows the Apache NetBeans IDE with a project named 'JDBC_GUI_Project'. The 'Main.java' file is open, displaying Java code for a DQL Select Query. The code includes comments and comments for creating a connection, creating a statement, and executing a select query. The code is as follows:

```
try {  
    // Load Driver.  
    Class.forName("com.mysql.cj.jdbc.Driver");  
  
    // Create Connection.  
    MyConn = DriverManager.getConnection(url, username, password);  
  
    // Create Statement.  
    MySt = MyConn.createStatement();  
  
    // Execute Selection query.  
    MyRs = MySt.executeQuery("select * from employee");  
  
    // Fetch the Table Data Row by Row as a ResultSet (How to Fetch Data).  
    while(MyRs.next()) {  
        empID = "Employee_ID = " + MyRs.getString(1) + "\n";  
        empName = "Full Name = " + MyRs.getString(2) + "\n";  
        empSalary = "Salary = " + MyRs.getString(3) + "\n";  
        empAddress = "Address = " + MyRs.getString(4) + "\n";  
        empCity = "City = " + MyRs.getString(5) + "\n";  
        empState = "State = " + MyRs.getString(6) + "\n";  
        empZip = "Zip = " + MyRs.getString(7) + "\n";  
        empInfo.append(empID + empName + empSalary + empAddress + empCity + empState + empZip + "\n");  
    }  
  
    // Close Connection.  
    MySt.close();  
    MyConn.close();  
}  
  
catch(SQLException ex) {  
    JOptionPane.showMessageDialog(null, "An Error Appeared In Your JDBC operation !", "JDBC Error", JOptionPane.ERROR_MESSAGE);  
}
```

Performing A DQL Select Query

Structure Of The Project



References

- 1) Telusko / Navin Reddy : *Practical JDBC*.
<https://www.youtube.com/watch?v=5vzCjvUwMXg>
- 2) Telusko : *JDBC Theory Tutorial*.
https://www.youtube.com/watch?v=y_YxwyYRJek
- 3) Telusko : *Setting Up MySQL Workbench*.
<https://www.youtube.com/watch?v=WDEyt2VHpj4>
- 4) Maurice Muteti : *Loading MySQL Connector/Java To Classpath*.
- 5) Stack Overflow : *Deprecated Driver Class "com.mysql.jdbc.Driver"* .
- 6) MySQL Documentation : *Changes In MySQL Connector/Java API*.
- 7) W3Schools.com : *SQL Tutorial*.