IBM DB2® 9.7

DB2 application development Hands-On Lab

Information Management Cloud Computing Center of Competence

IBM Canada Lab

Contents

CONTENTS 2

1. INTRODUCTION 3

2. OBJECTIVES 3

3. SUGGESTED READING 3

4. SETUP 3

4.1 Environment Setup Requirements 3
4.2 Login to the Virtual Machine 4

5. DEVELOPING STORED PROCEDURES, UDFS AND TRIGGERS 5

5.1 Procedure to create a SQL PL stored procedure 5
5.2 Procedure to create a UDF 7
5.3 Procedure to create a trigger 8

6. WORKING WITH A CLIENT APPLICATION TO ACCESS A DB2 DATABASE 12

6.1 Create and populate a table 12
6.2 Open the Application in IBM Data Studio 13
6.3 Install JDBC Driver 15
7. Connecting to DB2 19
7.1 Closing the Connection 20

8. QUERYING DATA 20

8.1 Incorporating SELECT with the Application 22
8.2 Search the Database using the Application 23

9. INSERTING DATA 26

9.1 Incorporating INSERT with the Application 27
9.2 Insert into the Database using the Application 28

1. Introduction

In this lab you will create a stored procedure, a user-defined function, and a trigger. Later in the lab you will work with a JDBC application that accesses a DB2 database.

2. Objectives

After completion of this lab, you should be able to:

- Develop stored procedures, UDFs and triggers
- Install the JDBC driver in IBM Data Studio
- Write Java code that can:
    - Create a connection to DB2
    - Properly close a connection to DB2
    - Query data using SELECT statements
    - Add new data to the database

3. Suggested reading

Getting started with DB2 Application Development eBook (Chapter 1, 3, 4)
https://www.ibm.com/developerworks/wikis/display/db2oncampus/FREE+ebook+-+Getting+started+with+DB2+application+development

A free eBook that can quickly get you up to speed with DB2 application development

4. Setup

4.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

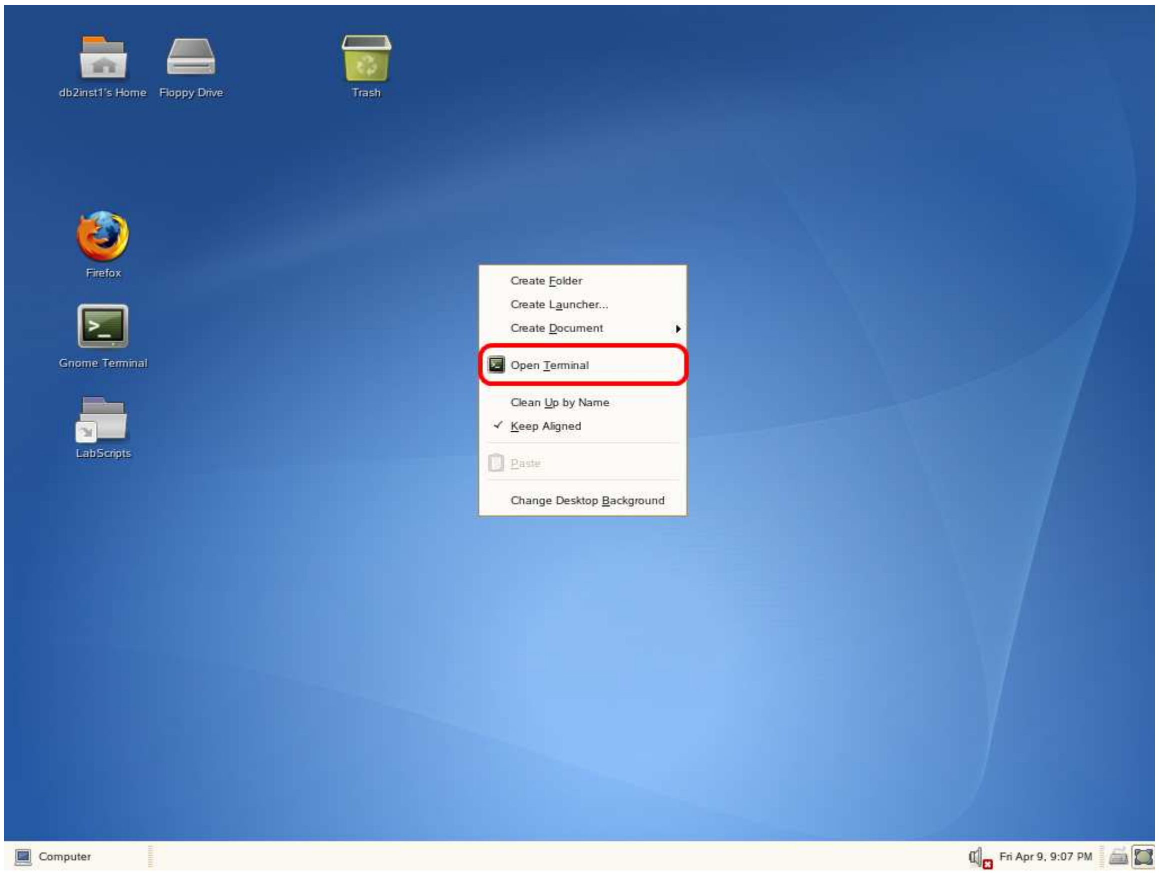
For help on how to obtain these components please follow the instructions specified in the **VMware Basics and Introduction** module.

4.2 Login to the Virtual Machine

1. Login to the VMware virtual machine using the following information:
   User: **db2inst1**
   Password: **password**

2. Type in the command **"startx**" to bring up the graphical environment.

3. Open a terminal window as by right-clicking on the **Desktop** area and choose the "**Open Terminal**" item.



4. Start up DB2 Server by typing "**db2start**" in the terminal window.

```
db2start
```

5. For executing this lab, you will need the DB2's sample database created in its original format. Execute the commands below to drop (if it already exists) and recreate the SAMPLE database:

```
db2 force applications all
db2 drop db sample
db2sampl
```

5. Developing stored procedures, UDFs and Triggers

5.1 Procedure to create a SQL PL stored procedure

1. Let's create a SQL PL Stored Procedure using IBM Data Studio. IBM Data Studio is not required to develop stored procedures; however, we recommend this tool as it makes development easier.

2. Open IBM Data Studio

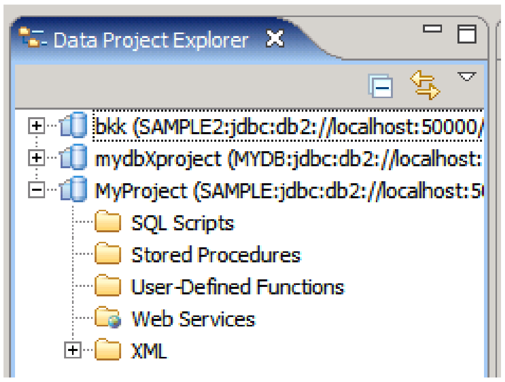
3. Click on the Data Project Explorer window, then choose:
   `File → New → Project → Data Development Project`

4. After clicking next, give the name *MyProject* to your project, and click *Next*

5. Choose the database you want to associate this project with from the list, then click *Next,* and *Finish.* If you get a dialog box indicating your profile is not complete, click on *Edit.* Choose *Driver properties* and ensure your *User name* and *password* fields are completed, and that clicking on *Test Connection* works. If all is OK, then click *OK* and *Finish*.

6. The database name should be SAMPLE. The rest of default values should be correct. Provide your *user ID/psw* and click on *Test Connection*. If the connection is successful, click on *Next*, and then on *Finish*.

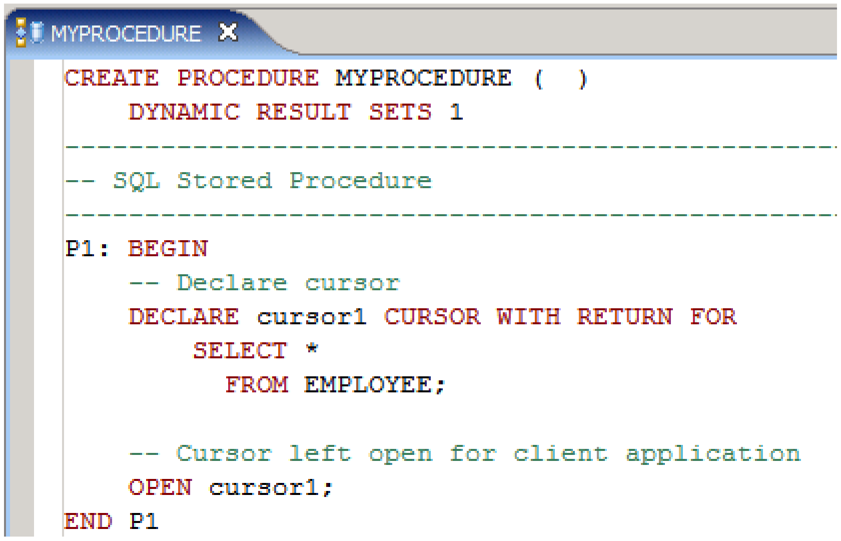
7. On the Data Project Explorer you should now be able to see your project. Drill down your project tree as shown below to list folders for SQL Scripts, Stored Procedures, etc.


Data Project Explorer
bkk (SAMPLE2:jdbc:db2://localhost:50000/
mydbXproject (MYDB:jdbc:db2://localhost:
MyProject (SAMPLE:jdbc:db2://localhost:5
SQL Scripts
Stored Procedures
User-Defined Functions
Web Services
XML

8. Right-click the Stored Procedures folder, and choose *New* → *Stored Procedure*

9. Provide a name for your stored procedure, for example use *myprocedure*, then click *Next*.

10. In the SQL Statements window, IBM Data Studio provides you with a sample statement *SELECT PROCSCHEMA, PROCNAME FROM SYSCAT.PROCEDURES*. Replace this statement with: *SELECT * FROM EMPLOYEE*, and click on *Finish*. You will then be presented with a template code as shown below.



```
CREATE PROCEDURE MYPROCEDURE (  )
    DYNAMIC RESULT SETS 1
------------------------------------------------
-- SQL Stored Procedure
------------------------------------------------
P1: BEGIN
    -- Declare cursor
    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT *
          FROM EMPLOYEE;

    -- Cursor left open for client application
    OPEN cursor1;
END P1
```

11. The code provided by IBM Data Studio can be used as a template for you to write your own code. This code declares a cursor and then opens it at the end, which would mean that the cursor is to be returned to the caller. Without modification, let's use this stored procedure for illustration purposes.

12. In the Database Explorer panel, expand the tree to show your connection, database, schema, and stored procedures. Note the stored procedure *myprocedure* is not shown. The Database Explorer shows the objects in the database. At this moment, the *myprocedure* stored procedure is not in the database since it has not been deployed. To deploy the procedure, right-click on it from the Data Project Explorer window, and choose *Deploy*. On the Deploy options window, let's choose all default, and click on *Finish*.

13. If the deployment was not successful, review the error codes received at deployment time, and fix them. If the deployment is successful, you are ready to run the procedure. Right-click on the procedure from the Data Project Explorer panel, and choose *Run*. At the bottom right-corner you will see the results of the procedure.

14. Now that the procedure has been successfully coded, deployed and run, you can test running it from the DB2 CLP:

```
db2 => connect to sample
db2 => call myprocedure()
```

5.2 Procedure to create a UDF

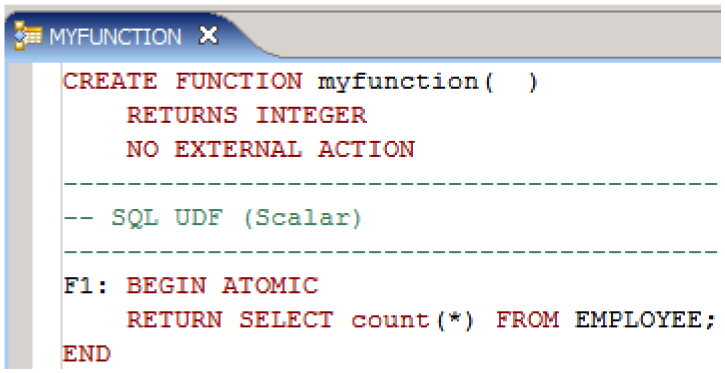
1. Creating user-defined functions (UDFs), is very similar to creating Stored Procedures. You can create a UDF by issuing the CREATE FUNCTION statement from the DB2 Command Window or Linux shell. For ease of development, we recommend you to use IBM Data Studio.

2. To develop a UDF from IBM Data Studio, let's use the same project we used when working with Stored Procedures. Right-click the User-Defined Functions folder, and click *New → User-Defined Function* menu item.

3. Give the name *myfunction* to your function, and click on *Next*.

4. In the SQL Statement or Expression window, IBM Data Studio will provide an SQL statement as example. Change this statement to:
   SELECT count(*) FROM EMPLOYEE

5. At this point click on *finish*. A template as shown in the figure below should be displayed:



```
CREATE FUNCTION myfunction(  )
    RETURNS INTEGER
    NO EXTERNAL ACTION
-----------------------------------------
-- SQL UDF (Scalar)
-----------------------------------------
F1: BEGIN ATOMIC
    RETURN SELECT count(*) FROM EMPLOYEE;
END
```

6. Let's now deploy the function and run it. From the Data Project Explorer panel, right-click on the function *myfunction*, and choose *Deploy*. On the Deploy options window, leave all defaults as they are, and click on *Finish*.

7. After a successful deployment, right-click again on the function, and choose *Run*. At the bottom right corner you will see the result.

8. To run the function from the DB2 CLP issue the following:

```
db2 => connect to sample
db2 => values myfunction()
```

9. Instead of using the values clause, you can also issue a SELECT statement as shown below:

```
db2 => connect to sample
db2 => select myfunction() from sysibm.sysdummy1
```

SYSIBM.SYSDUMMY1 is a table provided with DB2 that contains one row and one column. It can be used for this purpose where you are calling a function from a SELECT statement, but you don't really need to access any table. Because the SELECT statement requires a table as part of its syntax, we need to provide a table. Of course, you can create your own "dummy" table for this same purpose.

5.3 Procedure to create a trigger

1. You can create a trigger by issuing the CREATE TRIGGER statement from the DB2 CLP. If you would like to use IBM Data Studio, you have to create the trigger as a script. There is no folder specific for developing Triggers from Data Studio. The Control Center, which is

deprecated, can be used to create triggers. Start the Control Center from the Command Window or Linux shell issuing the command:

```
db2cc
```

2. For this example with triggers, you will need to create an additional table that will be used for logging as follows:

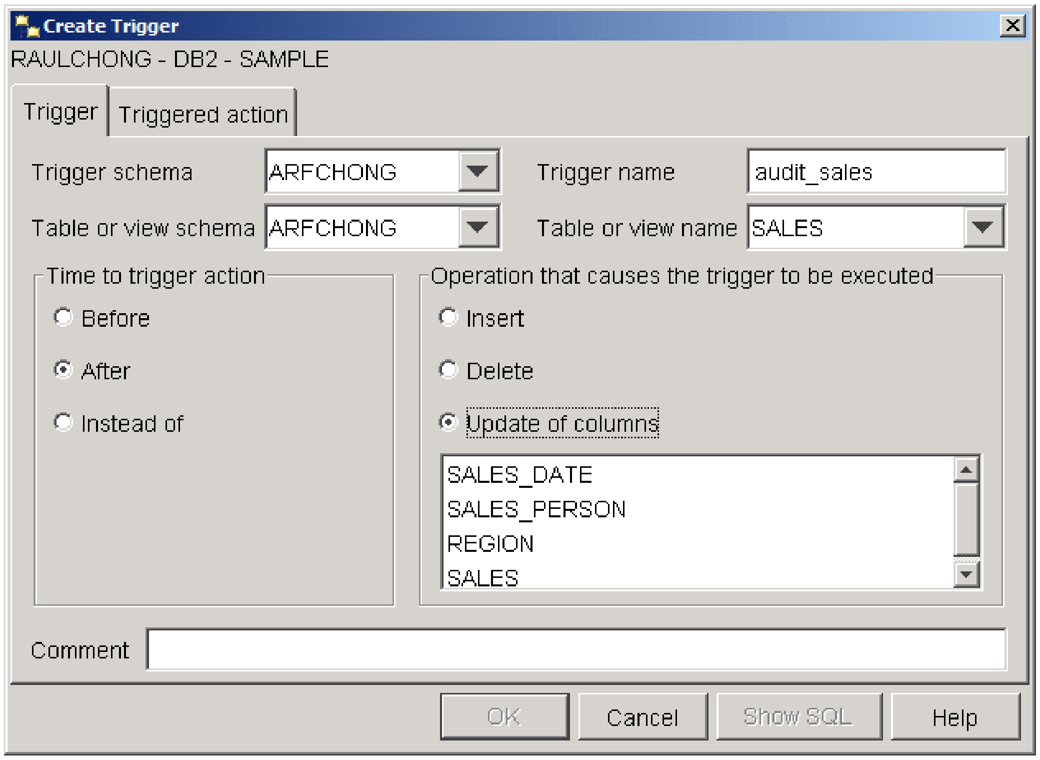
```
db2 => create table saleslog (userid varchar(128) not null,
daytime timestamp not null)
```

3. From the Control Center, expand the SAMPLE database folder. Right-click on the Triggers folder and select the *Create* option. The Create Trigger dialog window opens.

4. Fill in the following information in the dialog window:

Trigger property	**Value**
Trigger Schema	User ID of the user you are logged in as (should be the default setting)
Trigger Name	audit_sales
Table/View Schema	User ID of the user you are logged in as (should be the default setting)
Table/View Name	Sales
Time to trigger action	After
Operation that causes the trigger to be executed	Update of columns (do not specify any columns because we want the trigger to fire when any of the columns are updated).
Comment	Logs all update actions on Sales table.

5. The following figure shows the corresponding values to set:



Trigger schema: ARFCHONG
Trigger name: audit_sales
Table or view schema: ARFCHONG
Table or view name: SALES

Time to trigger action: Before [✓ After] Instead of

Operation that causes the trigger to be executed: Insert Delete [✓ Update of columns]

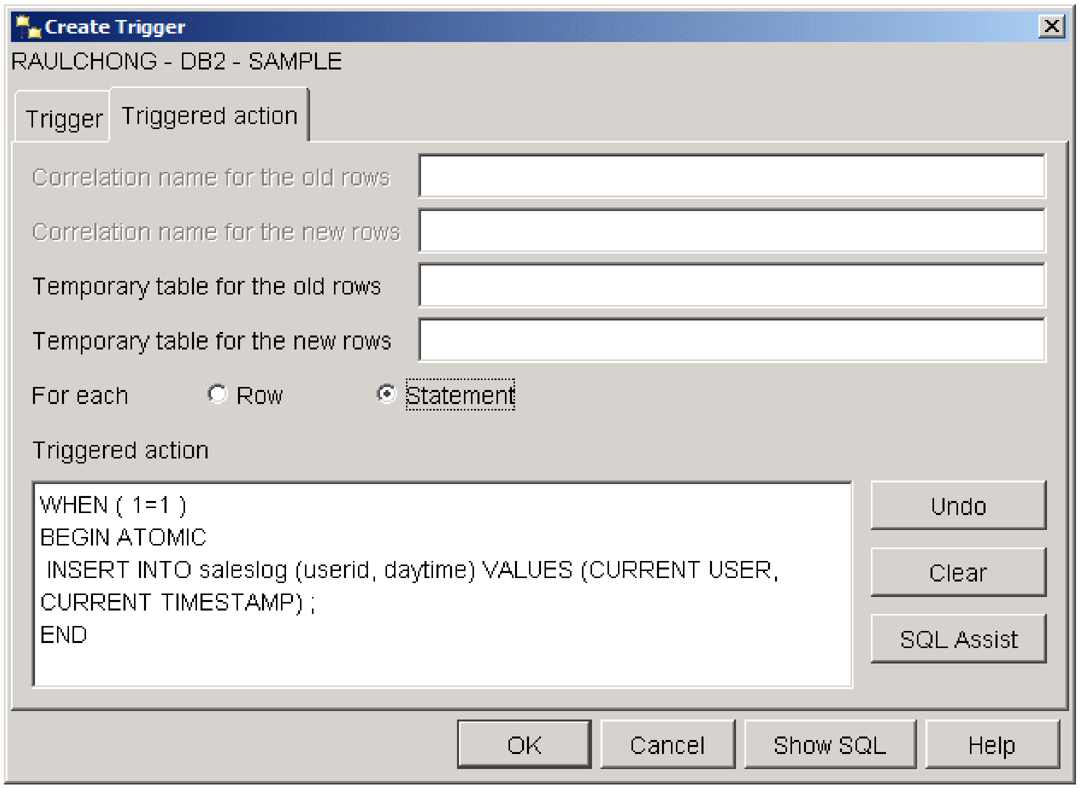
SALES_DATE
SALES_PERSON
REGION
SALES

Comment:

6. On the Triggered action tab, select the For Each STATEMENT option. Use the following code for the triggered action:

```
WHEN ( 1=1 )
BEGIN ATOMIC
   INSERT INTO saleslog (userid, daytime)
          VALUES (CURRENT USER, CURRENT TIMESTAMP);
 END
```

Note:
A statement trigger fires once after the statement activating the trigger has completed. This is specified with the FOR EACH STATEMENT option. A row trigger specifies that the triggered action will execute every time the triggering SQL statement affects a row. This is specified with the FOR EACH ROW statement.



RAULCHONG - DB2 - SAMPLE

Correlation name for the old rows

Correlation name for the new rows

Temporary table for the old rows

Temporary table for the new rows

For each ( ) Row [✓ Statement]

Triggered action

```
WHEN ( 1=1 )
BEGIN ATOMIC
 INSERT INTO saleslog (userid, daytime) VALUES (CURRENT USER,
CURRENT TIMESTAMP) ;
END
```

Click the OK button to create the trigger.

7. You should now be able to view the trigger in the Triggers folder in Control Center.

8. Query the saleslog table to ensure there is no data in it. Delete any rows that may be in it using DELETE FROM saleslog

```
db2 => delete from saleslog
```

9. Update a record in the sales table to test the trigger:

```
db2 => update sales set sales_date = current date where year
(sales_date) = 1996
```

10. Check the contents of the saleslog table. How many rows are in it?

6. Working with a client application to access a DB2 database

In this part of the lab, you will work with a JDBC application that will access a DB2 database. You first need to create and populate a table used for this exercise, and configure the JDBC in Data Studio.

6.1 Create and populate a table

We will create a simple table that will be updated during this lab session. The table named “**ESQLEMPLOYEE**” will be created and will be populated with 1 row of data.

1. Change to the directory where the script files are.

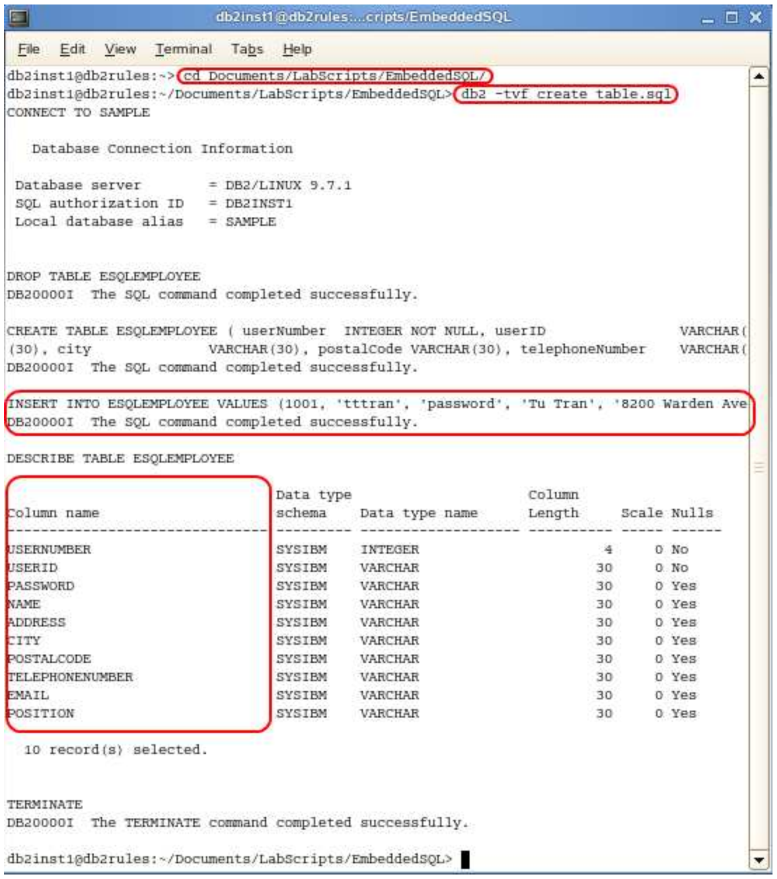
```
cd /home/db2inst1/Documents/LabScripts/EmbeddedSQL
```

2. We will take a look at the simple query first by using the command

```
cat create_table.sql
```

3. To run the query, in the terminal window, type in

```
db2 –tvf create_table.sql
```

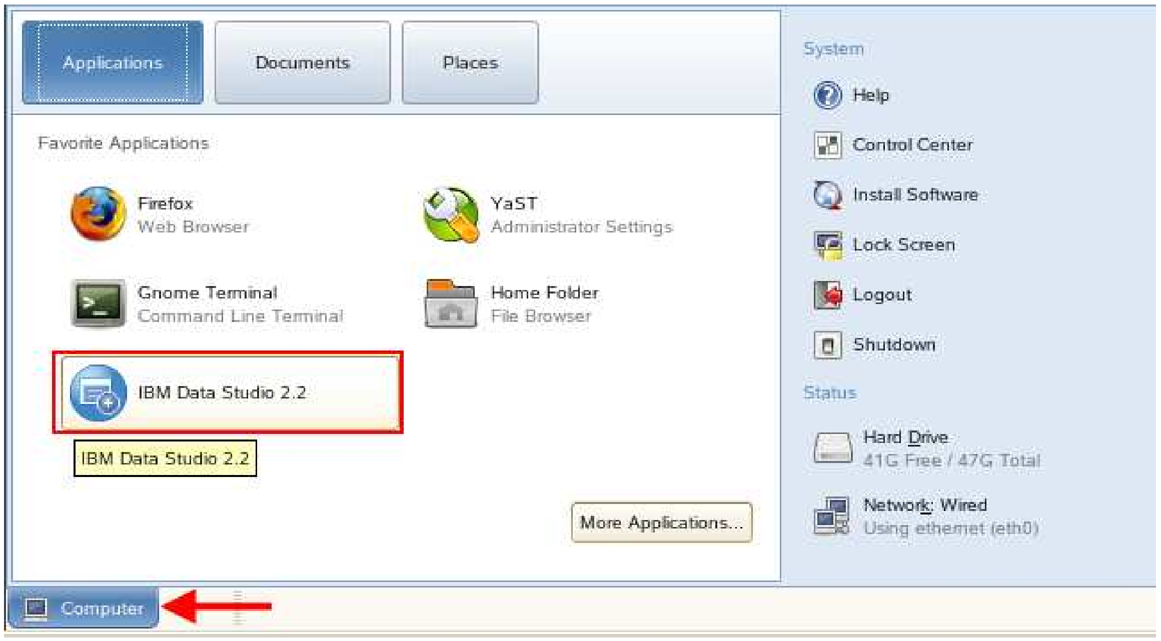

db2inst1@db2rules:...cripts/EmbeddedSQL
File Edit View Terminal Tabs Help

```
db2inst1@db2rules:~> cd Documents/LabScripts/EmbeddedSQL/
db2inst1@db2rules:~/Documents/LabScripts/EmbeddedSQL> db2 -tvf create_table.sql
CONNECT TO SAMPLE

   Database Connection Information

 Database server        = DB2/LINUX 9.7.1
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE


DROP TABLE ESQLEMPLOYEE
DB20000I  The SQL command completed successfully.

CREATE TABLE ESQLEMPLOYEE ( userNumber  INTEGER NOT NULL, userID              VARCHAR(
(30), city              VARCHAR(30), postalCode VARCHAR(30), telephoneNumber    VARCHAR(
DB20000I  The SQL command completed successfully.

INSERT INTO ESQLEMPLOYEE VALUES (1001, 'tttran', 'password', 'Tu Tran', '8200 Warden Ave
DB20000I  The SQL command completed successfully.

DESCRIBE TABLE ESQLEMPLOYEE

                                Data type                     Column
Column name                     schema    Data type name      Length     Scale Nulls
------------------------------- --------- ------------------- ---------- ----- ------
USERNUMBER                      SYSIBM    INTEGER                      4     0 No
USERID                          SYSIBM    VARCHAR                     30     0 No
PASSWORD                        SYSIBM    VARCHAR                     30     0 Yes
NAME                            SYSIBM    VARCHAR                     30     0 Yes
ADDRESS                         SYSIBM    VARCHAR                     30     0 Yes
CITY                            SYSIBM    VARCHAR                     30     0 Yes
POSTALCODE                      SYSIBM    VARCHAR                     30     0 Yes
TELEPHONENUMBER                 SYSIBM    VARCHAR                     30     0 Yes
EMAIL                           SYSIBM    VARCHAR                     30     0 Yes
POSITION                        SYSIBM    VARCHAR                     30     0 Yes

  10 record(s) selected.


TERMINATE
DB20000I  The TERMINATE command completed successfully.

db2inst1@db2rules:~/Documents/LabScripts/EmbeddedSQL> 
```

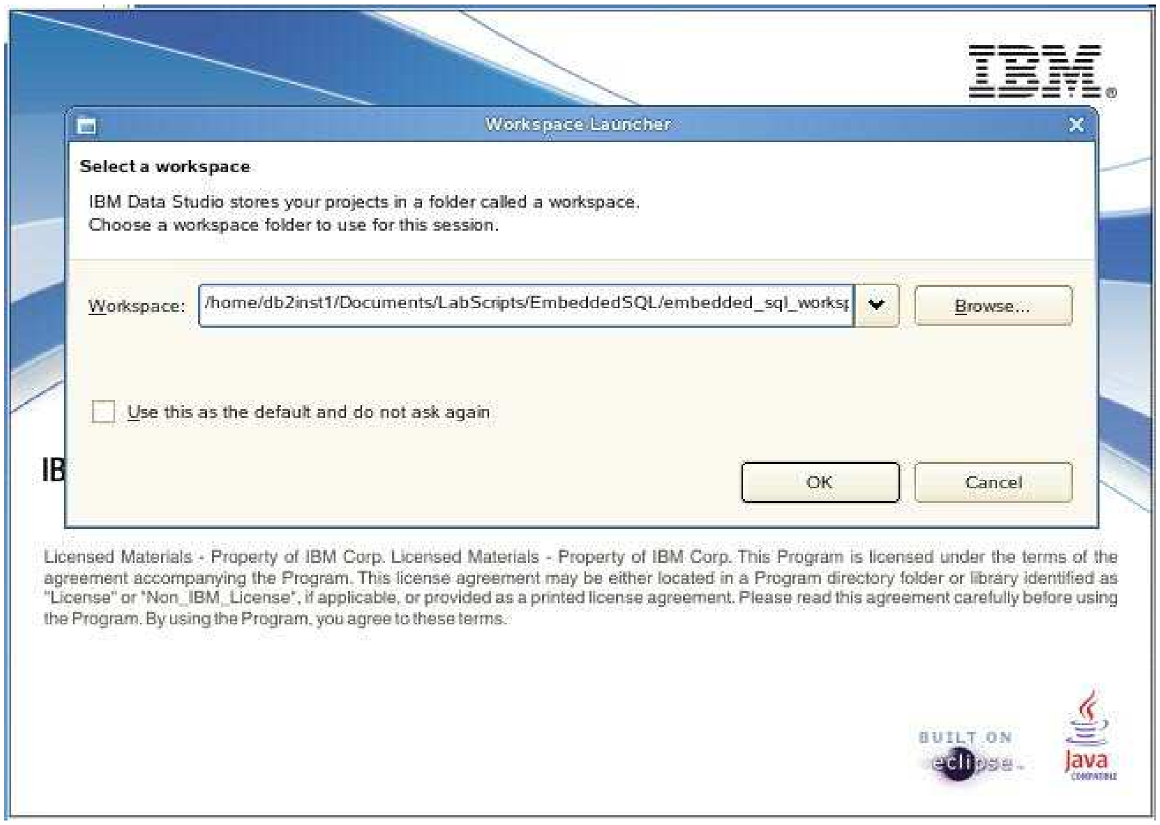
6.2 Open the Application in IBM Data Studio

Now that the database is ready and the ESQLEMPLOYEE table is created, we need to open the application that we will be working with in IBM Data Studio. Once opened, we can begin configuring the DB2 JDBC driver for the application.

1. Open IBM Data Studio by clicking **Computer** and choosing **IBM Data Studio 2.2.**



2. A prompt to select a workstation will appear. Enter "`/home/db2inst1/Documents/LabScripts/EmbeddedSQL/embedded_sql_workspace`" as the workstation and select **OK**.



Select a workspace

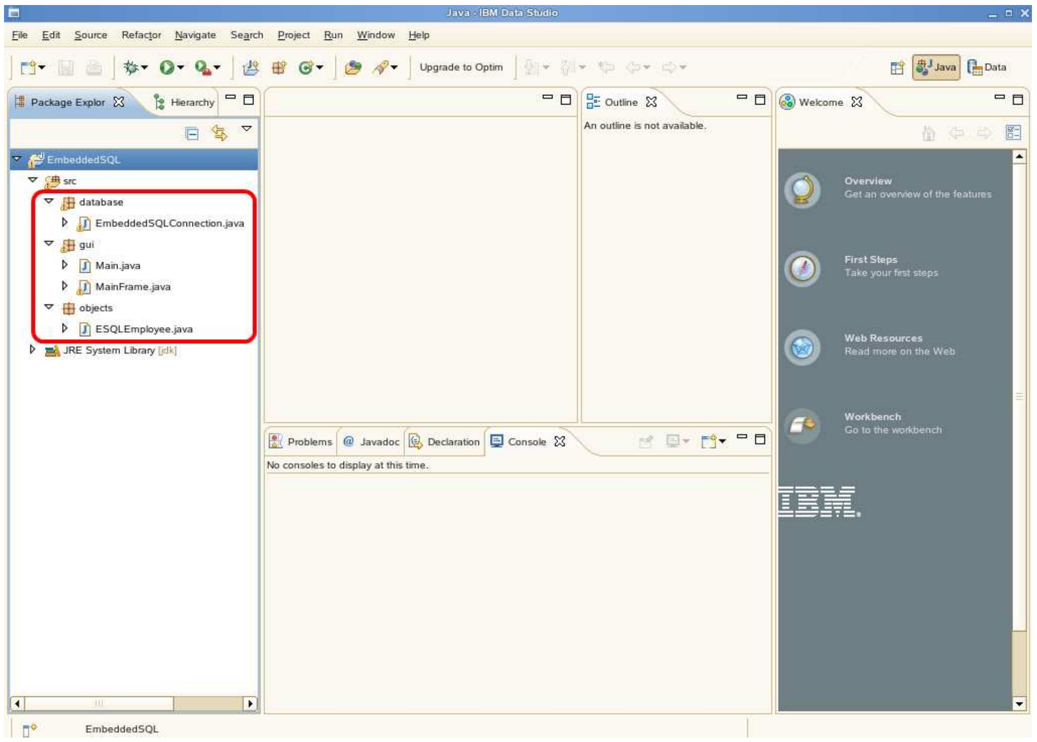
IBM Data Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/db2inst1/Documents/LabScripts/EmbeddedSQL/embedded_sql_worksp

☐ Use this as the default and do not ask again

Licensed Materials - Property of IBM Corp. Licensed Materials - Property of IBM Corp. This Program is licensed under the terms of the agreement accompanying the Program. This license agreement may be either located in a Program directory folder or library identified as "License" or "Non_IBM_License", if applicable, or provided as a printed license agreement. Please read this agreement carefully before using the Program. By using the Program, you agree to these terms.

3. The following screen will be displayed.

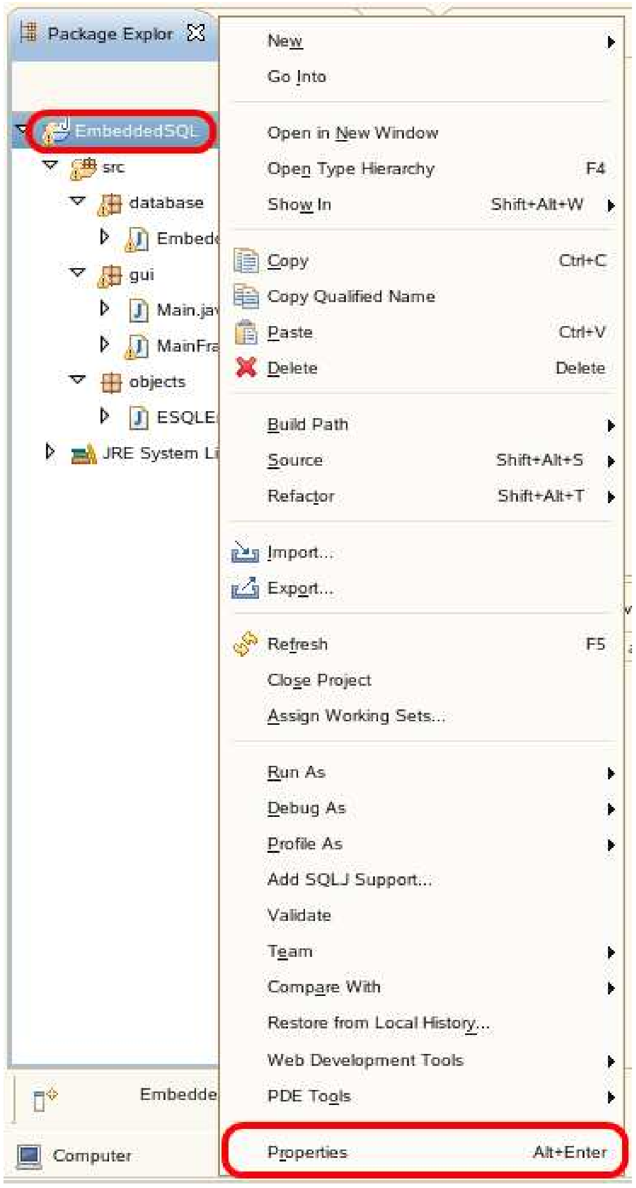


We have successfully opened our application in IBM Data Studio; however it is not ready to connect to DB2 just yet. In order to connect to DB2 we must first install the JDBC driver in our project.

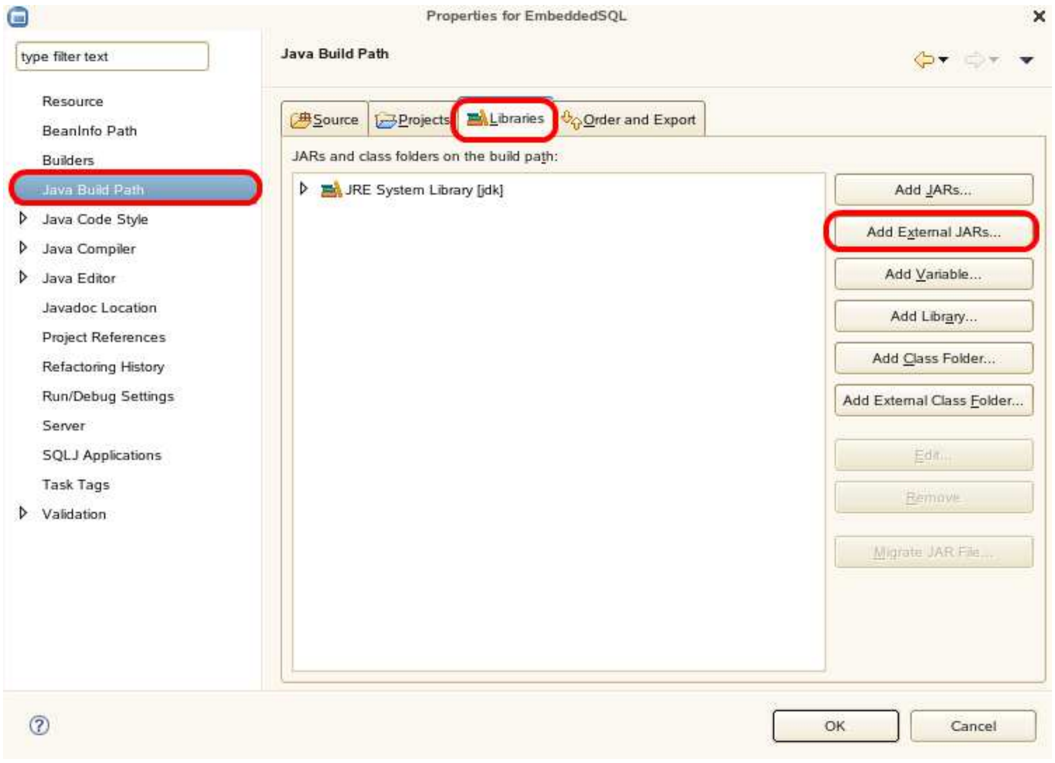
6.3 Install JDBC Driver

The JDBC Driver allows Java applications to connect to SQL compliant databases, send SQL statements, and process return messages and data.

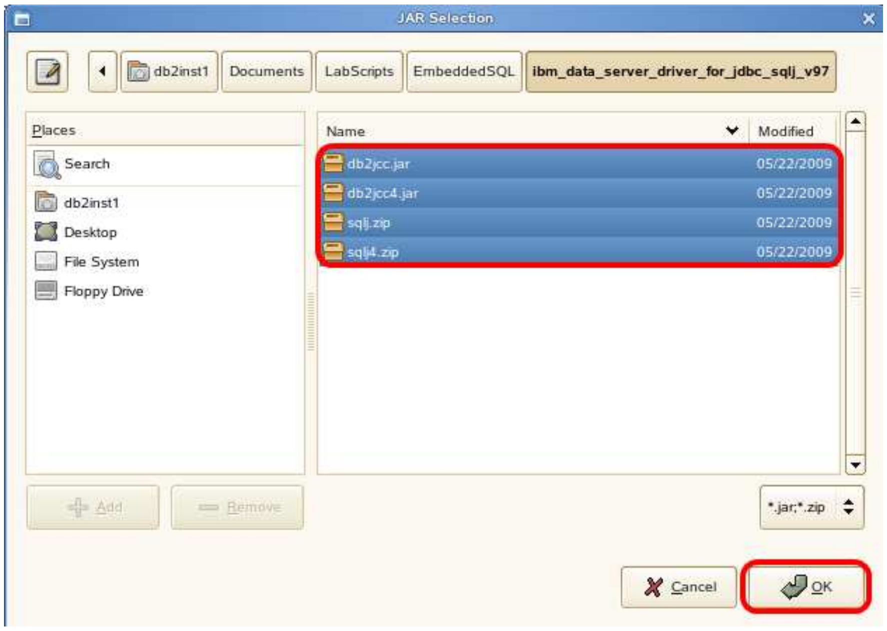
1. With IBM Data Studio opened, right click **EmbeddedSQL** and select **Properties.**



2. Select **Java Build Path** from the list. Then select the **Libraries** tab and click the **Add External JARs** button.

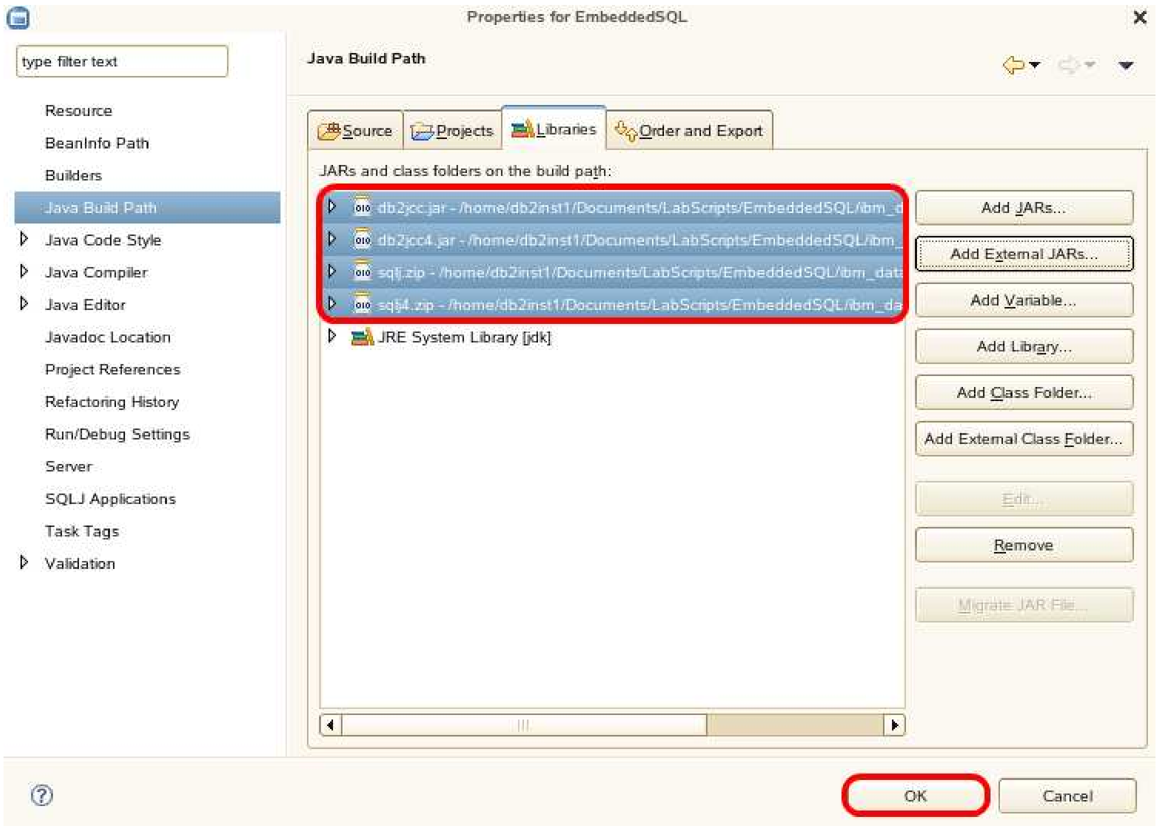


3. Browse to `“/home/db2inst1/Documents/LabScripts/EmbeddedSQL/ibm_data_server_driver_for_jdbc_sqlj_v97”` and select all the files in this folder.

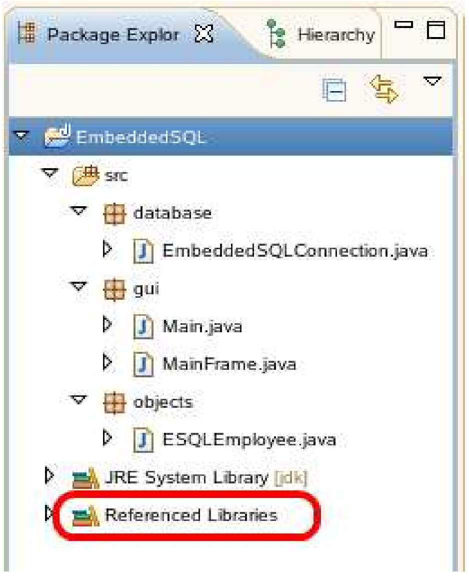


Name	Modified
db2jcc.jar	05/22/2009
db2jcc4.jar	05/22/2009
sqlj.zip	05/22/2009
sqlj4.zip	05/22/2009

4. The selected files will now appear under the **Libraries Tab**. Select **OK** to continue.



5. The **JDBC Driver** has been successfully installed. You can view the libraries that were added by selecting **Referenced Libraries** in the **Package Explorer**.

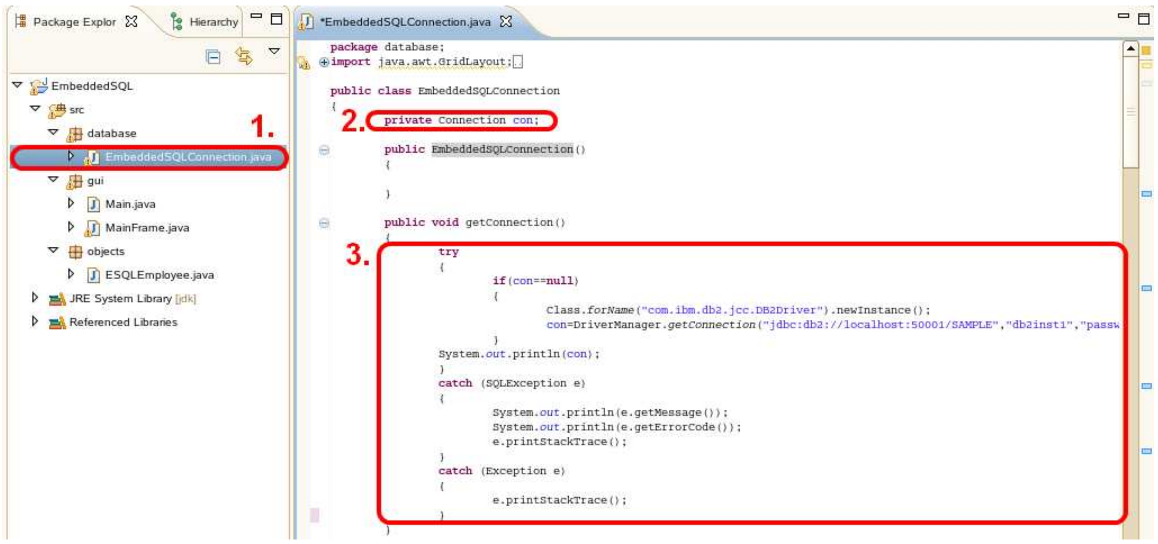


Now that the JDBC driver is properly set up in our development environment, we can start adding code in our application to connect to DB2.

7. Connecting to DB2

Whenever we wish to interact with a database, we must first establish a connection to the database server.

1. In the database package, open the EmbeddedSQLConnection class. Select the EmbeddedSQL project and press F5 to open the class



```
package database;
import java.awt.GridLayout;

public class EmbeddedSQLConnection
{
        private Connection con;

        public EmbeddedSQLConnection()
        {

        }

        public void getConnection()
        {
                try
                {
                        if(con==null)
                        {
                                Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
                                con=DriverManager.getConnection("jdbc:db2://localhost:50001/SAMPLE","db2inst1","passw
                        }
                System.out.println(con);
                }
                catch (SQLException e)
                {
                        System.out.println(e.getMessage());
                        System.out.println(e.getErrorCode());
                        e.printStackTrace();
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
```

The EmbeddedSQLConnection class will be the most important class in this exercise. This class contains all the functions in the application related to data access.

2. In order to create a connection, a variable of type Connection must first be declared to hold the Connection object.

```
private Connection con;
```

Now we can create the connection to the DB2 server using the DriverManager object, and finally we store that connection in the “con” variable.

3. Complete the getConnection() function by uncommenting the code provided within getConnection().

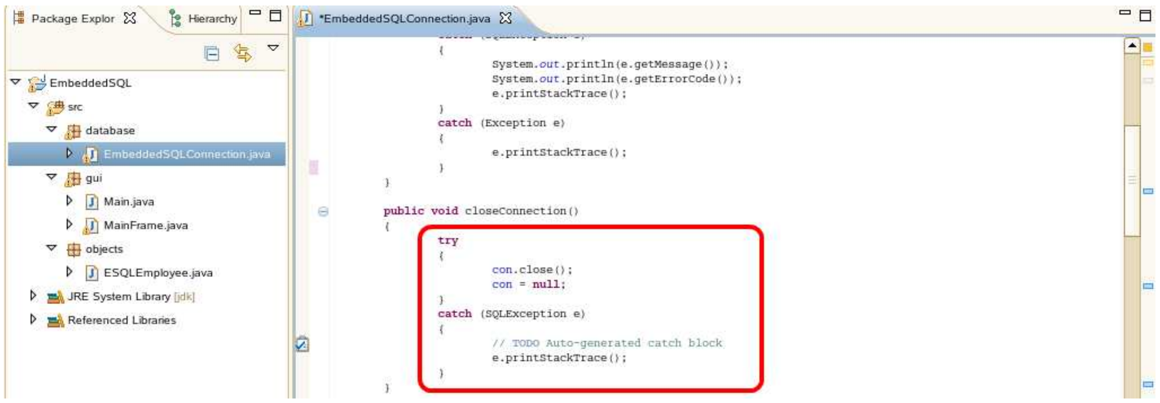
```
con=DriverManager.getConnection("jdbc:db2://localhost:50001/SAMPLE","db
2inst1","password");
```

The method getConnection() is attempting to establish a connection to the given database URL. We are connecting to DB2’s SAMPLE database with the user “db2inst1” using the password “password” through the JDBC API on port 50001.

7.1 Closing the Connection

Now that we know how to establish a connection, we also need to know how to properly close our connection to DB2 once it is not needed any more. This is an important step as it will free up system's resources for your application and the database server.

1. In the EmbeddedSQLConnection class, complete the closeConnection() function by uncommenting the code provided within closeConnection().



```
        {
                System.out.println(e.getMessage());
                System.out.println(e.getErrorCode());
                e.printStackTrace();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
}

public void closeConnection()
{
        try
        {
                con.close();
                con = null;
        }
        catch (SQLException e)
        {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}
```

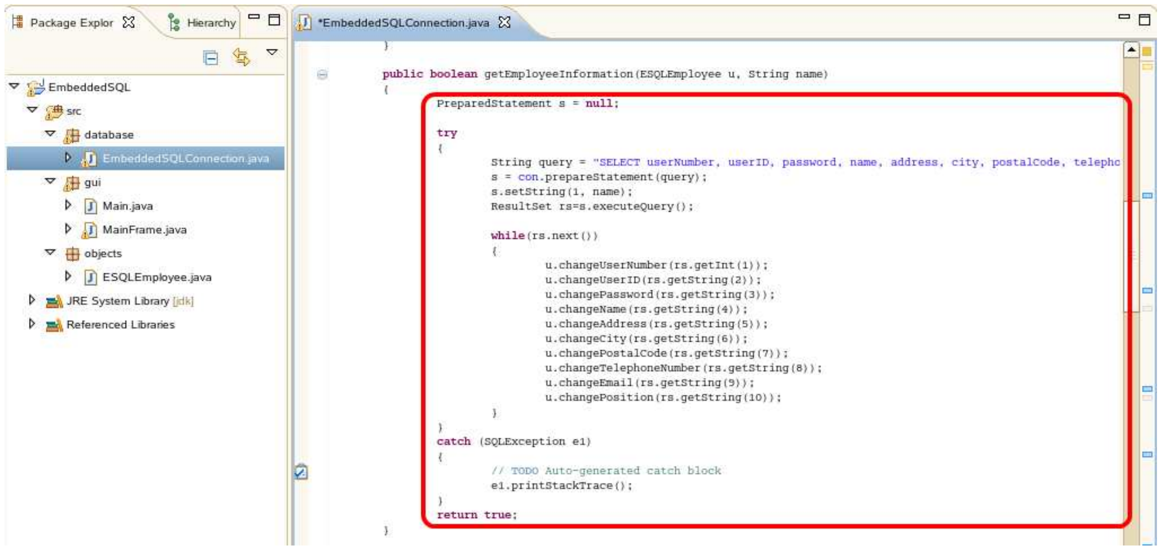
```
con.close()
```

The method close() is terminating the connection to the database specified during the getConnection() method above. Once the connection is terminated we can also set `con = null`.

8. Querying Data

Now that we have created functions to create and close a connection to DB2, we are ready to write a query to search through and display data.

1. In the EmbeddedSQLConnection class, complete the getEmployeeInformation() function by uncommenting the code provided within getEmployeeInformation ().



```
	}

	public boolean getEmployeeInformation(ESQLEmployee u, String name)
	{
		PreparedStatement s = null;

		try
		{
			String query = "SELECT userNumber, userID, password, name, address, city, postalCode, telepho
			s = con.prepareStatement(query);
			s.setString(1, name);
			ResultSet rs=s.executeQuery();

			while(rs.next())
			{
				u.changeUserNumber(rs.getInt(1));
				u.changeUserID(rs.getString(2));
				u.changePassword(rs.getString(3));
				u.changeName(rs.getString(4));
				u.changeAddress(rs.getString(5));
				u.changeCity(rs.getString(6));
				u.changePostalCode(rs.getString(7));
				u.changeTelephoneNumber(rs.getString(8));
				u.changeEmail(rs.getString(9));
				u.changePosition(rs.getString(10));
			}
		}
		catch (SQLException e1)
		{
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		return true;
	}
```

The getEmployeeInformation() function selects employee information based on a provided name. It is used by the application to search through the database for a specific employee.

2. In order to create and execute a query, an object of type PreparedStatement must first be specified.

```
PreparedStatement s = null;
```

The PreparedStatement object "s", will be used to hold the SQL SELECT statement.

3. The SQL statements itself is coded as a String.

```
String query = "SELECT userNumber, userID, password, name, address,
city, postalCode, telephoneNumber, email, position FROM ESQLEMPLOYEE
WHERE name = ?";
```

4. We can now create the PreparedStatement using the Connection object `con` and the prepareStatement() method. The resulting object is stored in the variable "s".

```
s = con.prepareStatement(query);
```

5. Looking at the query String, notice the "?". This is called a parameter marker. It marks the place where a value will be inserted during runtime, in this case, the search criteria provided by the user when executing the application.
The following command is used to associate a value with the parameter marker.

```
s.setString(1, name);
```

For example: If we wish to search for an employee with the name Tu Tran, the String query becomes, `"SELECT userNumber, userID, password, name, address, city, postalCode, telephoneNumber, email, position FROM ESQLEMPLOYEE WHERE name = Tu Tran";`.

6. As we execute the query, the results returned from the query must be stored. We store the data in an object of the type ResultSet.

```
ResultSet rs=s.executeQuery();
```

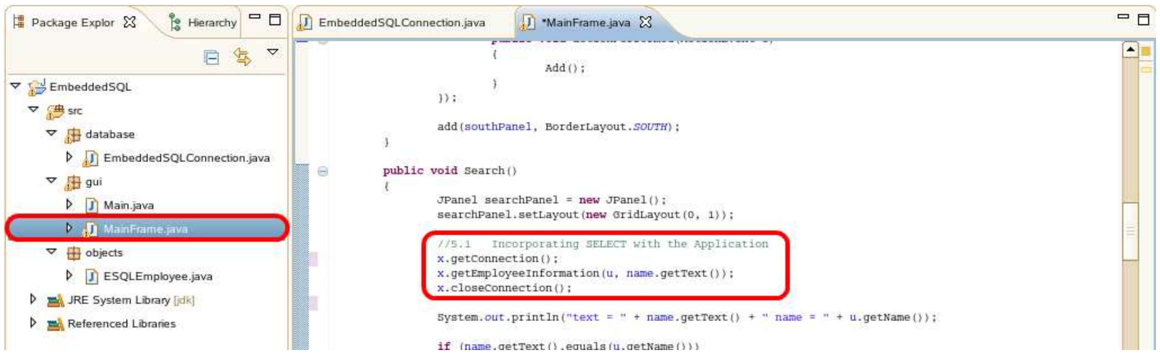
7. Finally we can retrieve the data stored in the ResultSet.

```
while(rs.next())
{

      u.changeUserNumber(rs.getInt(1));

      u.changeUserID(rs.getString(2));

      u.changePassword(rs.getString(3));

      u.changeName(rs.getString(4));

      u.changeAddress(rs.getString(5));

      u.changeCity(rs.getString(6));

      u.changePostalCode(rs.getString(7));

      u.changeTelephoneNumber(rs.getString(8));

      u.changeEmail(rs.getString(9));

      u.changePosition(rs.getString(10));

}
```

8.1 Incorporating SELECT with the Application

We have created functions to create and close a connection to DB2 as well as to return data using a SELECT statement. How can we use these functions in our application?

1. In the GUI package, open the MainFrame class.



```
                {
                        Add();
                }
            });

            add(southPanel, BorderLayout.SOUTH);
    }

    public void Search()
    {
            JPanel searchPanel = new JPanel();
            searchPanel.setLayout(new GridLayout(0, 1));

            //5.1   Incorporating SELECT with the Application
            x.getConnection();
            x.getEmployeeInformation(u, name.getText());
            x.closeConnection();

            System.out.println("text = " + name.getText() + " name = " + u.getName());

            if (name.getText().equals(u.getName()))
```

The MainFrame class is where we will be using the functions created to interact with DB2. This class contains all the functions in the application that are responsible for allowing the user to interact with the application.

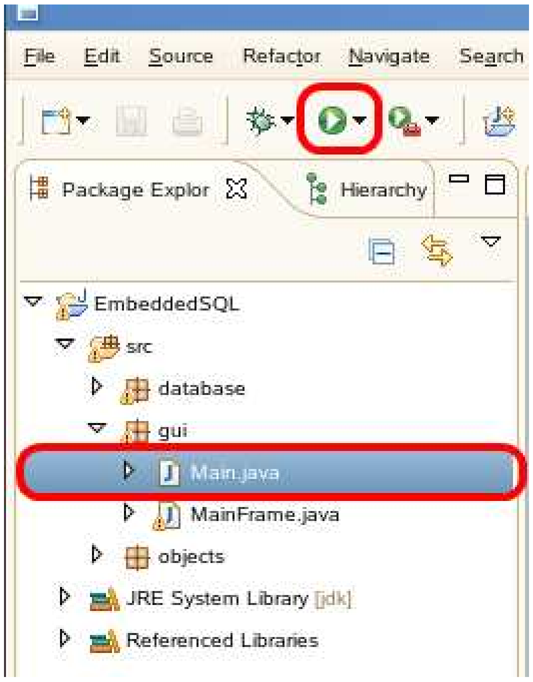
2. Go to the Search() function and uncomment the code provided.

```
//5.1 Incorporating SELECT with the Application
x.getConnection();
x.getEmployeeInformation(u, name.getText());
x.closeConnection();
```

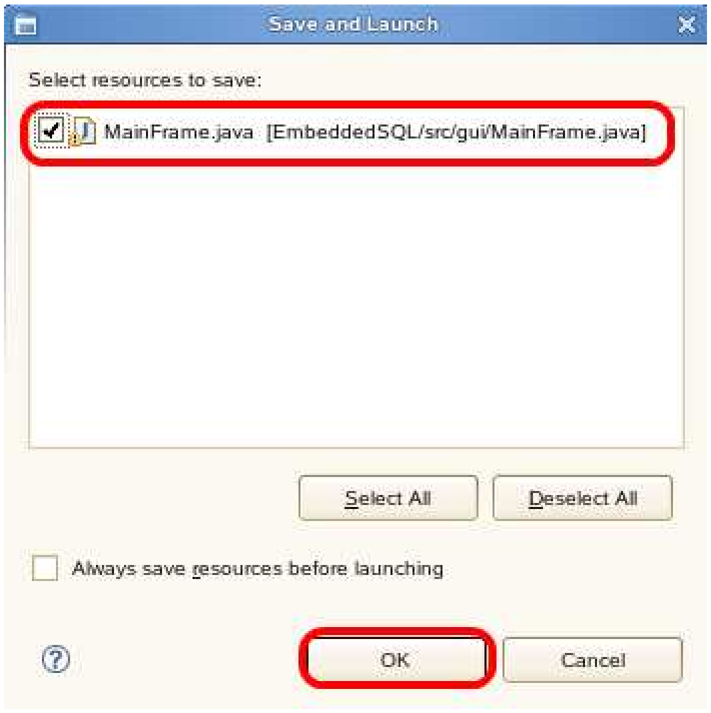
8.2 Search the Database using the Application

The application is now capable of performing a SELECT statement on the database and displaying the information returned.

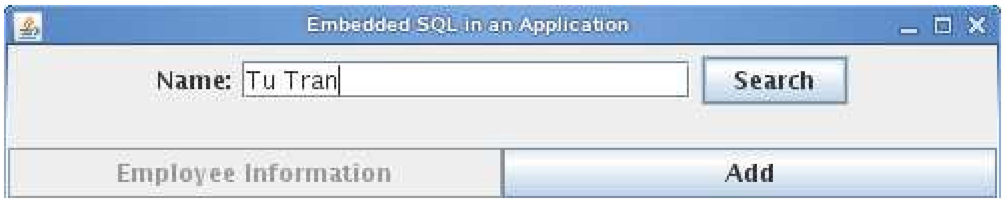
1. In the gui package, open the Main class and press .

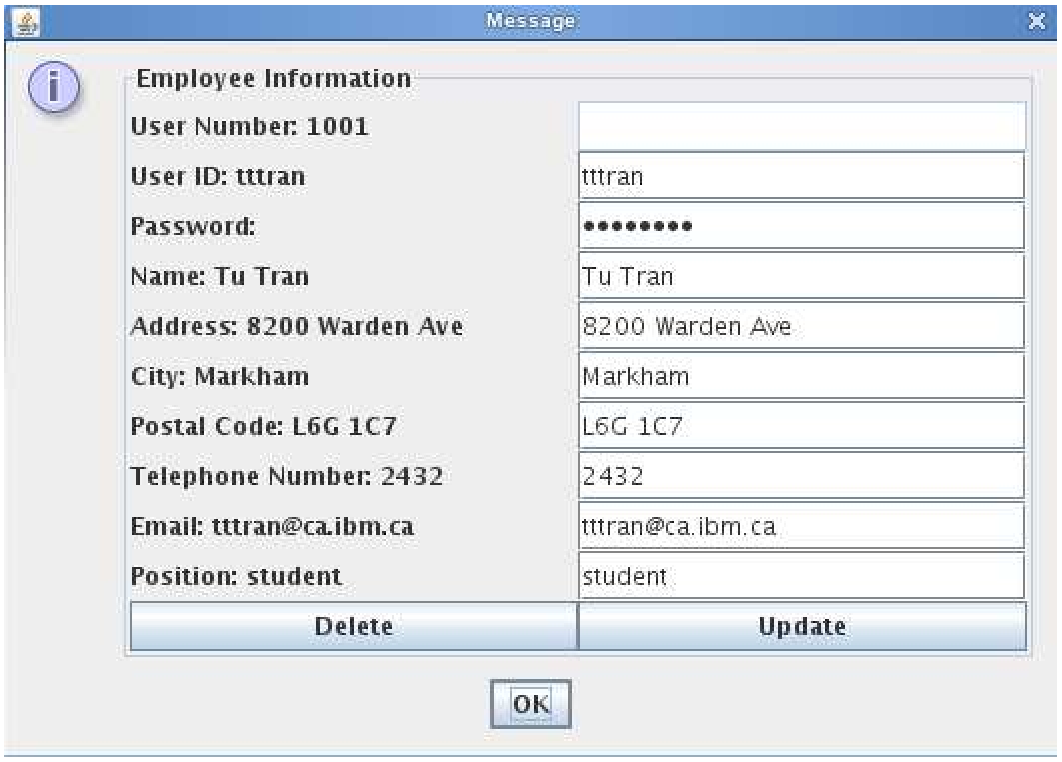


2. If you have not saved your changes, you will be prompted to save the file(s). Select all the resources that need to be saved and press **OK**.


Save and Launch
Select resources to save:
MainFrame.java [EmbeddedSQL/src/gui/MainFrame.java]
Select All
Deselect All
Always save resources before launching
OK
Cancel

3. The following program will appear. Enter the name “Tu Tran” and press Search.





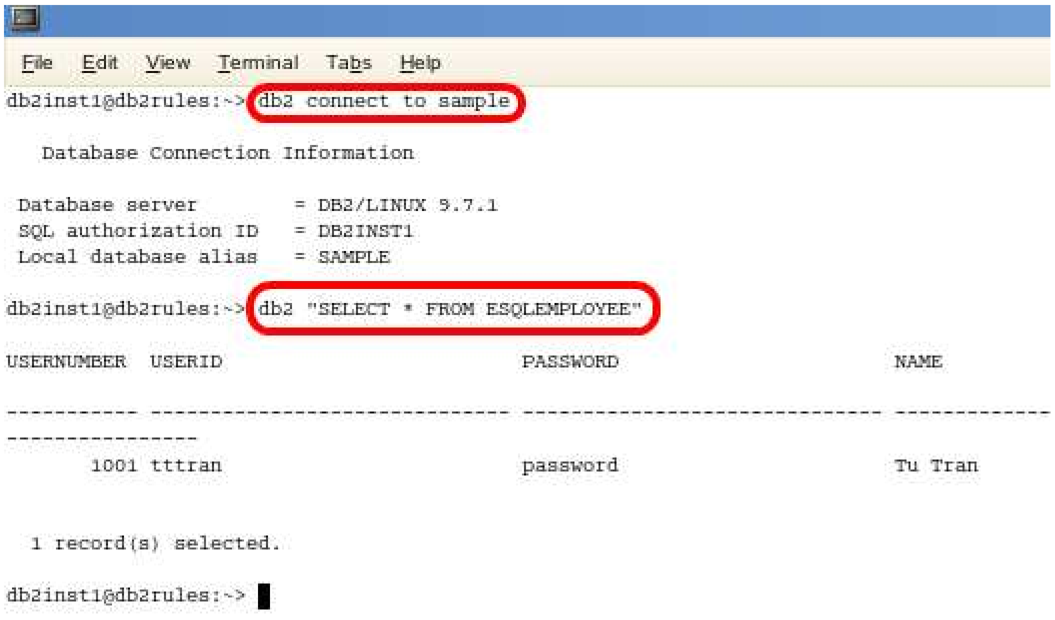
Employee Information

User Number: 1001	
User ID: tttran	tttran
Password:	••••••••
Name: Tu Tran	Tu Tran
Address: 8200 Warden Ave	8200 Warden Ave
City: Markham	Markham
Postal Code: L6G 1C7	L6G 1C7
Telephone Number: 2432	2432
Email: tttran@ca.ibm.ca	tttran@ca.ibm.ca
Position: student	student
Delete	Update

We can see that the following employee data was returned from the database.

4. Open a Terminal and connect to the Sample database to view the ESQLEMPLOYEE table.

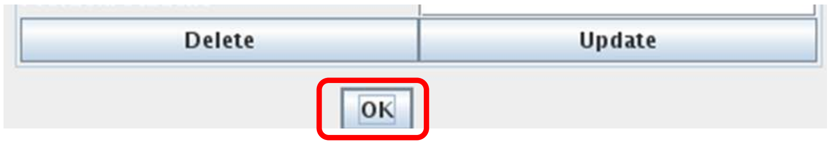
```
db2 connect to sample
db2 “SELECT * FROM ESQLEMPLOYEE”
```



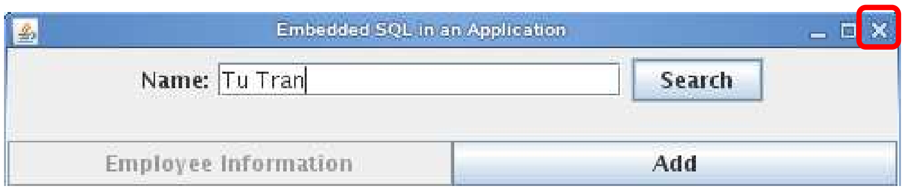
```
db2inst1@db2rules:~> db2 connect to sample

   Database Connection Information

 Database server        = DB2/LINUX 9.7.1
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

db2inst1@db2rules:~> db2 "SELECT * FROM ESQLEMPLOYEE"

USERNUMBER  USERID                         PASSWORD                       NAME

----------- ------------------------------ ------------------------------ -------------
----------------
       1001 tttran                         password                       Tu Tran

  1 record(s) selected.

db2inst1@db2rules:~>
```

We can see that the data returned through the Embedded SQL application is the same as the data we see by directly connecting to the Sample database.

5. Go back to the Java program and press “OK” to close the window.



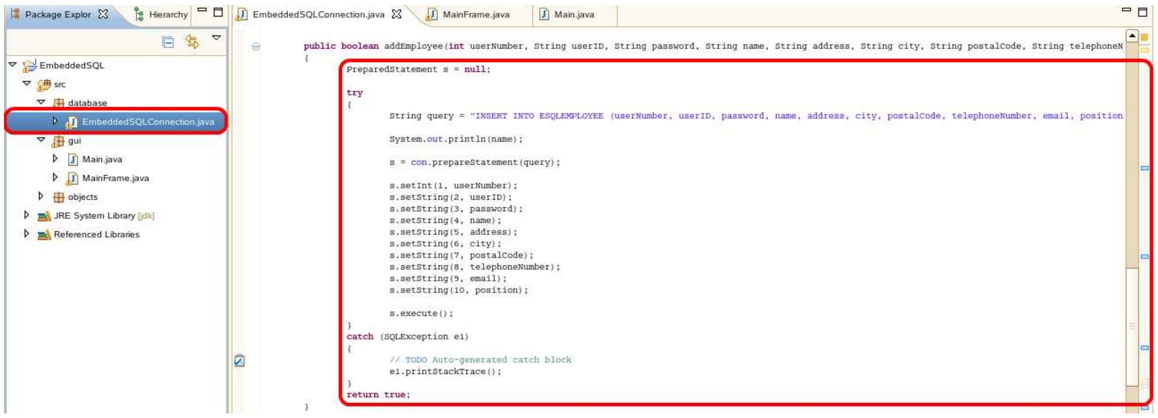
6. Press “X” to close the application.



9. Inserting Data

Using the INSERT statement is similar to using the SELECT statement except we do not need to store data in a ResultSet.

1. In the EmbeddedSQLConnection class, complete the addEmployee() function by uncommenting the code provided within addEmployee().



```
public boolean addEmployee(int userNumber, String userID, String password, String name, String address, String city, String postalCode, String telephoneN
{
    PreparedStatement s = null;

    try
    {
        String query = "INSERT INTO ESQLEMPLOYEE (userNumber, userID, password, name, address, city, postalCode, telephoneNumber, email, position

        System.out.println(name);

        s = con.prepareStatement(query);

        s.setInt(1, userNumber);
        s.setString(2, userID);
        s.setString(3, password);
        s.setString(4, name);
        s.setString(5, address);
        s.setString(6, city);
        s.setString(7, postalCode);
        s.setString(8, telephoneNumber);
        s.setString(9, email);
        s.setString(10, position);

        s.execute();
    }
    catch (SQLException e1)
    {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    return true;
}
```

The addEmployee () function inserts new employee information into the table ESQLEMPLOYEE.

2. As before, an object of type PreparedStatement is specified to store the SQL statement.

```
PreparedStatement s = null;
```

3. The SQL statement is coded as the following String.

```
String query = "INSERT INTO ESQLEMPLOYEE (userNumber, userID, password,
name, address, city, postalCode, telephoneNumber, email, position)
VALUES (?,?,?,?,?,?,?,?,?,?)";
```

4. We can now create the PreparedStatement using the Connection object `con` and the prepareStatement() method.

```
s = con.prepareStatement(query);
```

5. Notice the INSERT statement has several parameter markers. As before, they are necessary to associate values provided by the user to the SQL statement being executed.
   A command like the one below is used to associate a value to one of the parameters.

```
s.setString(2, userID);
```

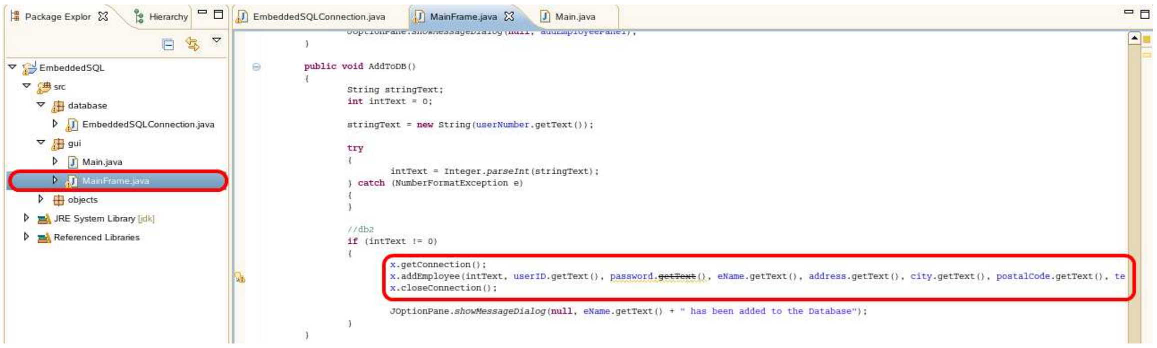
6. The statement can now be executed.

```
s.execute();
```

9.1 Incorporating INSERT with the Application

We have created functions to create and close a connection to DB2 as well insert data using an INSERT statement. How can we use these functions in our application?

1. In the GUI package, open the MainFrame class.



```
    }

    public void AddToDB()
    {
            String stringText;
            int intText = 0;

            stringText = new String(userNumber.getText());

            try
            {
                    intText = Integer.parseInt(stringText);
            } catch (NumberFormatException e)
            {
            }

            //db2
            if (intText != 0)
            {
                    x.getConnection();
                    x.addEmployee(intText, userID.getText(), password.~~getText~~(), eName.getText(), address.getText(), city.getText(), postalCode.getText(), te
                    x.closeConnection();

                    JOptionPane.showMessageDialog(null, eName.getText() + " has been added to the Database");
            }
    }
```

The MainFrame class is where we will be using the functions created to interact with DB2. This class contains all the functions in the application that are responsible for allowing the user to interact with the application.

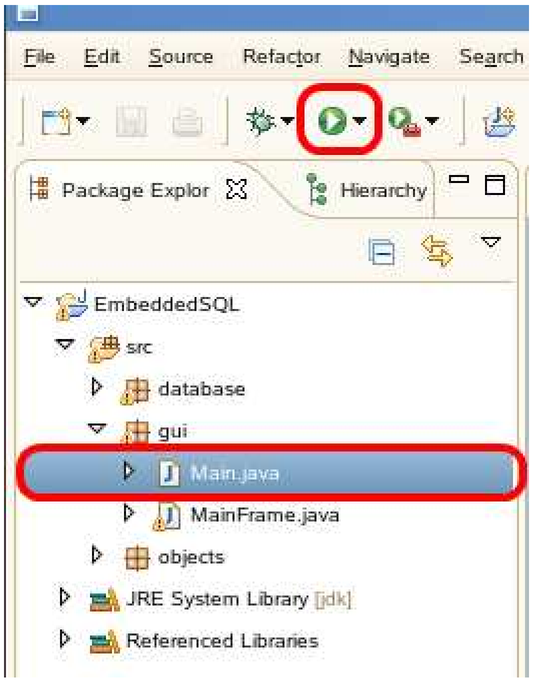
2. Go to the AddToDB() function and uncomment the code provided.

```
//6.1 Incorporating INSERT with the Application
x.getConnection();
x.addEmployee(intText, userID.getText(), password.getText(),
eName.getText(), address.getText(), city.getText(),
postalCode.getText(), telephoneNumber.getText(), email.getText(),
position.getText());
x.closeConnection();
```

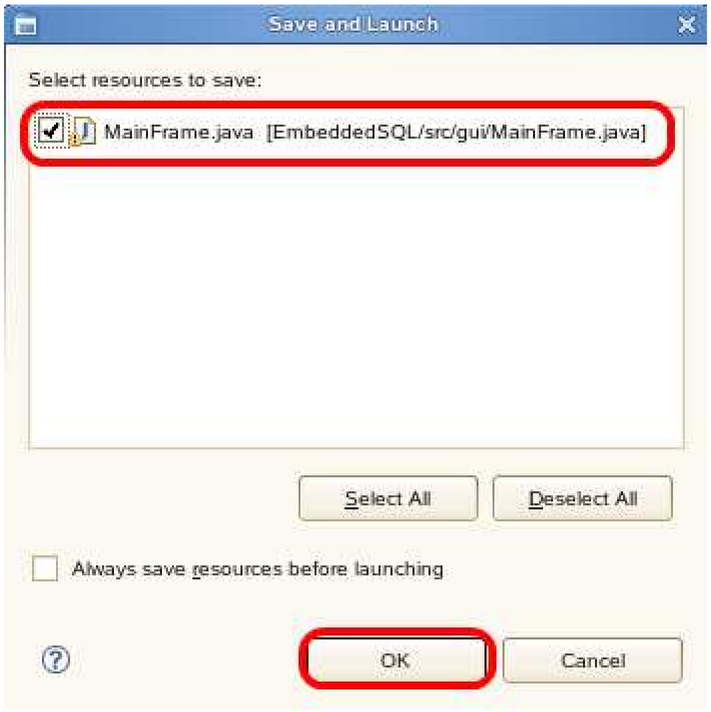
9.2 Insert into the Database using the Application

The application is now capable of performing an INSERT statement to add new employees to the database.

1. In the gui package, open the Main class and press ▶.



2. If you have not saved the changes, you will be prompted to save the file. Select all resources that need to be saved and press **OK**.


Save and Launch

Select resources to save:

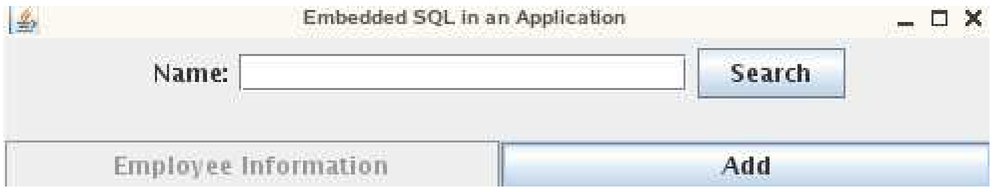
[✓] MainFrame.java [EmbeddedSQL/src/gui/MainFrame.java]

Select All | Deselect All

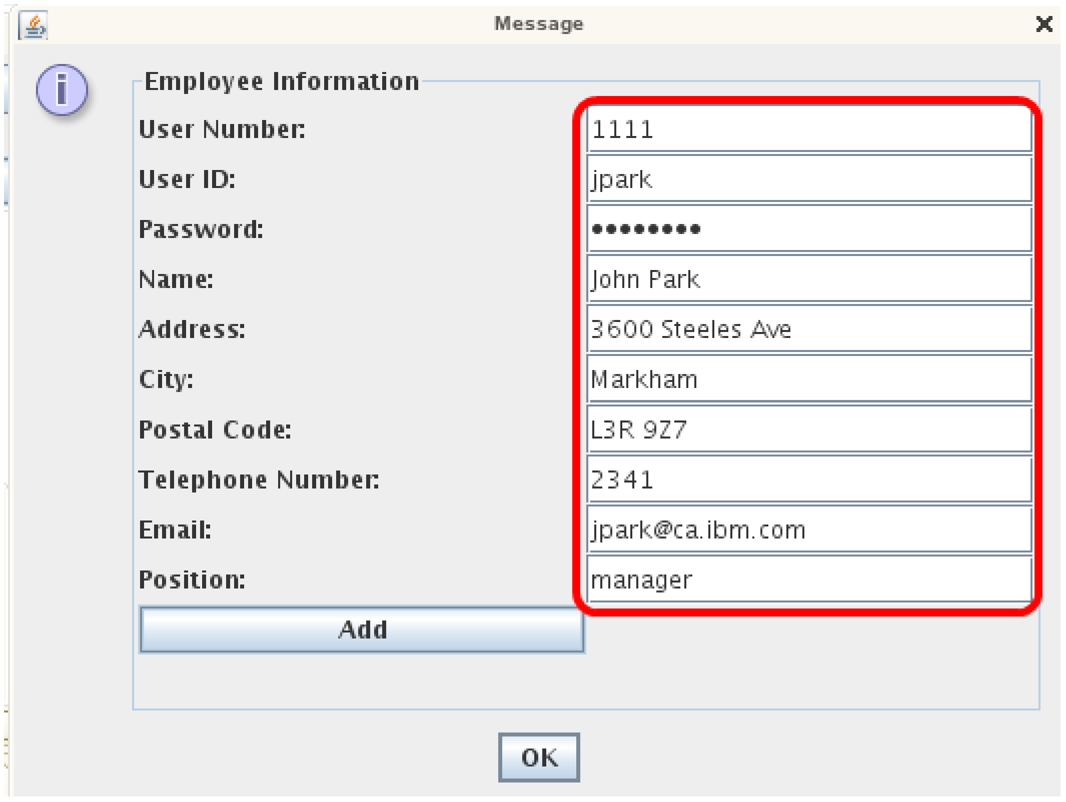
[ ] Always save resources before launching

OK | Cancel

3. The following program will appear. Press the Add button.

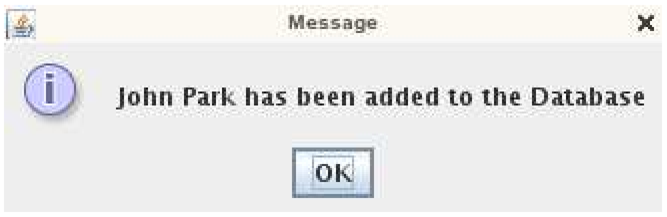


4. The following popup will appear. Enter the information as seen below and press Add.



User Number:	1111
User ID:	jpark
Password:	●●●●●●●●●
Name:	John Park
Address:	3600 Steeles Ave
City:	Markham
Postal Code:	L3R 9Z7
Telephone Number:	2341
Email:	jpark@ca.ibm.com
Position:	manager

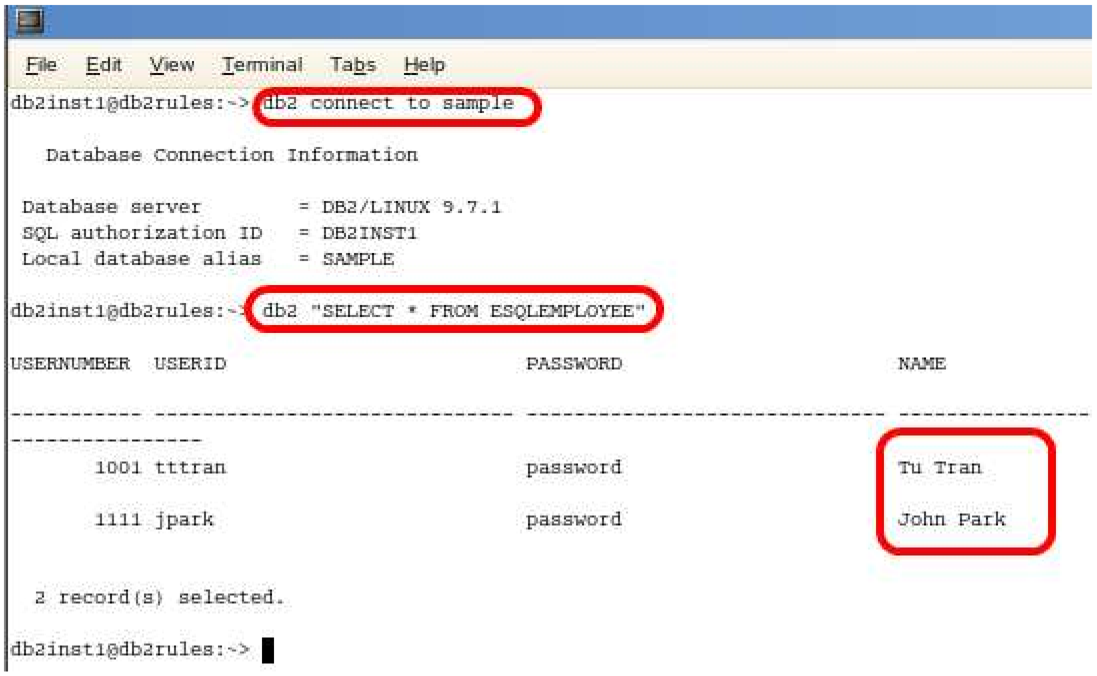
We can see that the employee data was successfully added to the database. Press "OK" to close the popup message.



John Park has been added to the Database

5. Open a Terminal and connect to the SAMPLE database to view the ESQLEMPLOYEE table.

```
db2 connect to sample
db2 "SELECT * FROM ESQLEMPLOYEE"
```



```
db2inst1@db2rules:~> db2 connect to sample

   Database Connection Information

 Database server        = DB2/LINUX 9.7.1
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

db2inst1@db2rules:~> db2 "SELECT * FROM ESQLEMPLOYEE"

USERNUMBER  USERID                          PASSWORD                        NAME

----------- ------------------------------ ------------------------------ ----------------
----------------
       1001 tttran                         password                        Tu Tran

       1111 jpark                          password                        John Park


  2 record(s) selected.

db2inst1@db2rules:~>
```

We can see that the employee “John Park” has been added to the Sample database.

Congratulations! You have just created a simple application capable of interacting with DB2. In this exercise we learned how to retrieve and insert data from a database using the JDBC API. Included with the application, there are also functions for the DELETE and UPDATE statements. Feel free to read through the code for a better understanding of how we can use JDBC within a Java application.