Lecture (15)
DB I

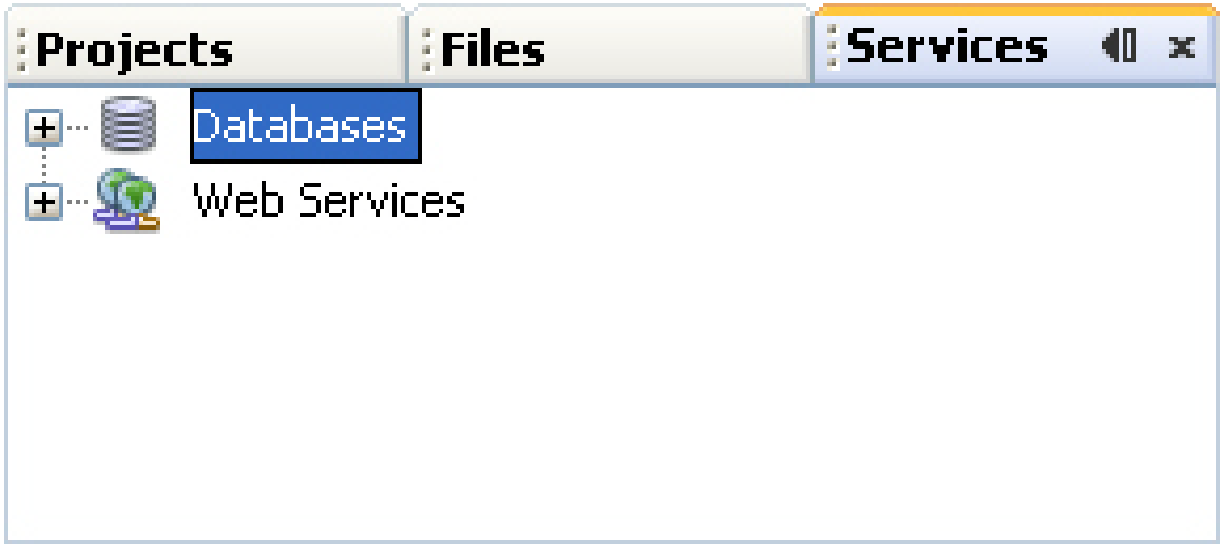
Dr. Ahmed ElShafee

Agenda

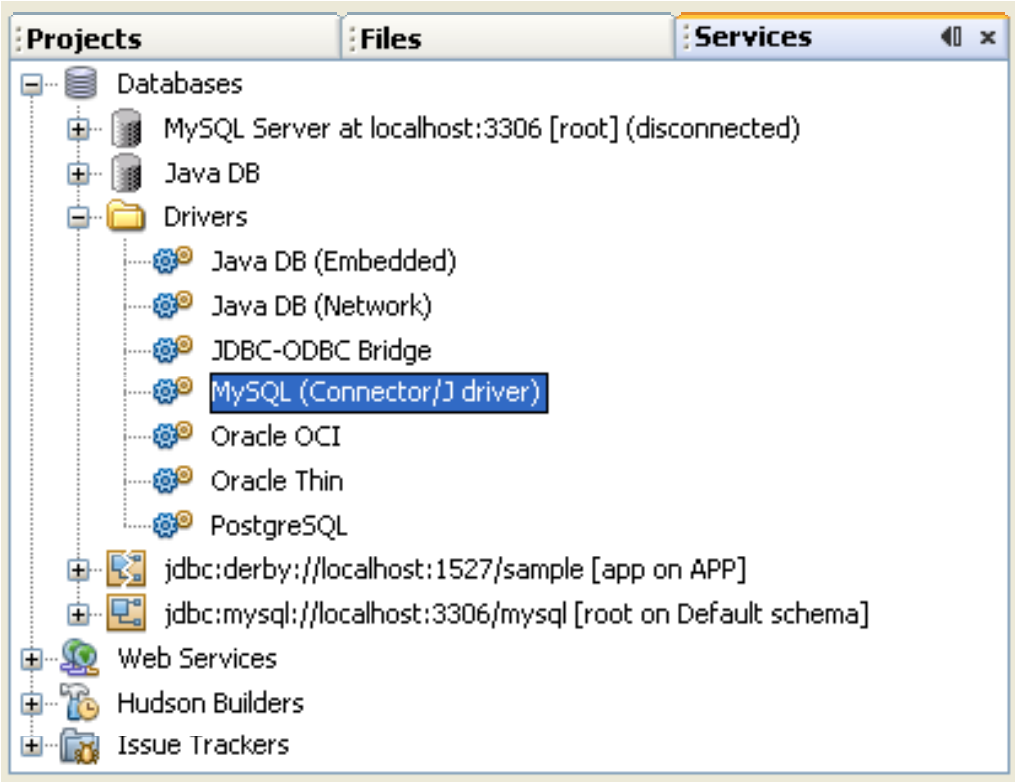
Introduction

- Java uses something called JDBC (Java Database Connectivity) to connect to databases.
- There's a JDBC API, which is the programming part, and a JDBC Driver Manager, which your programs use to connect to the database.
- JDBC allows you to connect to a wide-range of databases (Oracle, MySQL, etc),
- but we're going to use the in-built database you get with the Java/NetBeans software.
- The database is called Java DB, a version of Apache Derby.
- It runs on a virtual server, which you can stop and start from within NetBeans.

- To check that you have everything you need, have a look at the Services tab in NetBeans.
- If you can't see the Services tab, click Window from the NetBeans menu.
- From the Window menu, select Services.
- You should see something like this:

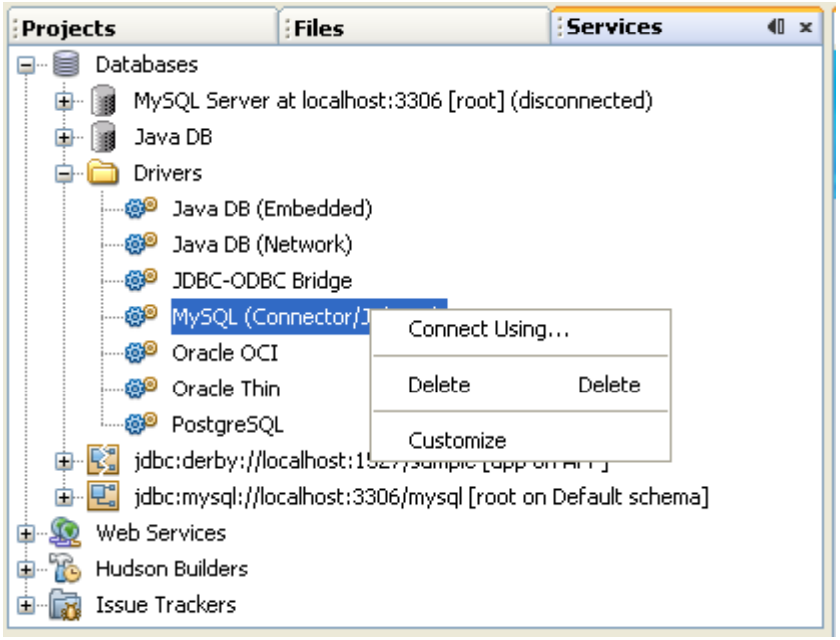


- Expand the Databases item to see a Java DB item, and a Drivers section:
- The idea is that you start the Java DB virtual server, and then create and manipulate databases on the server.

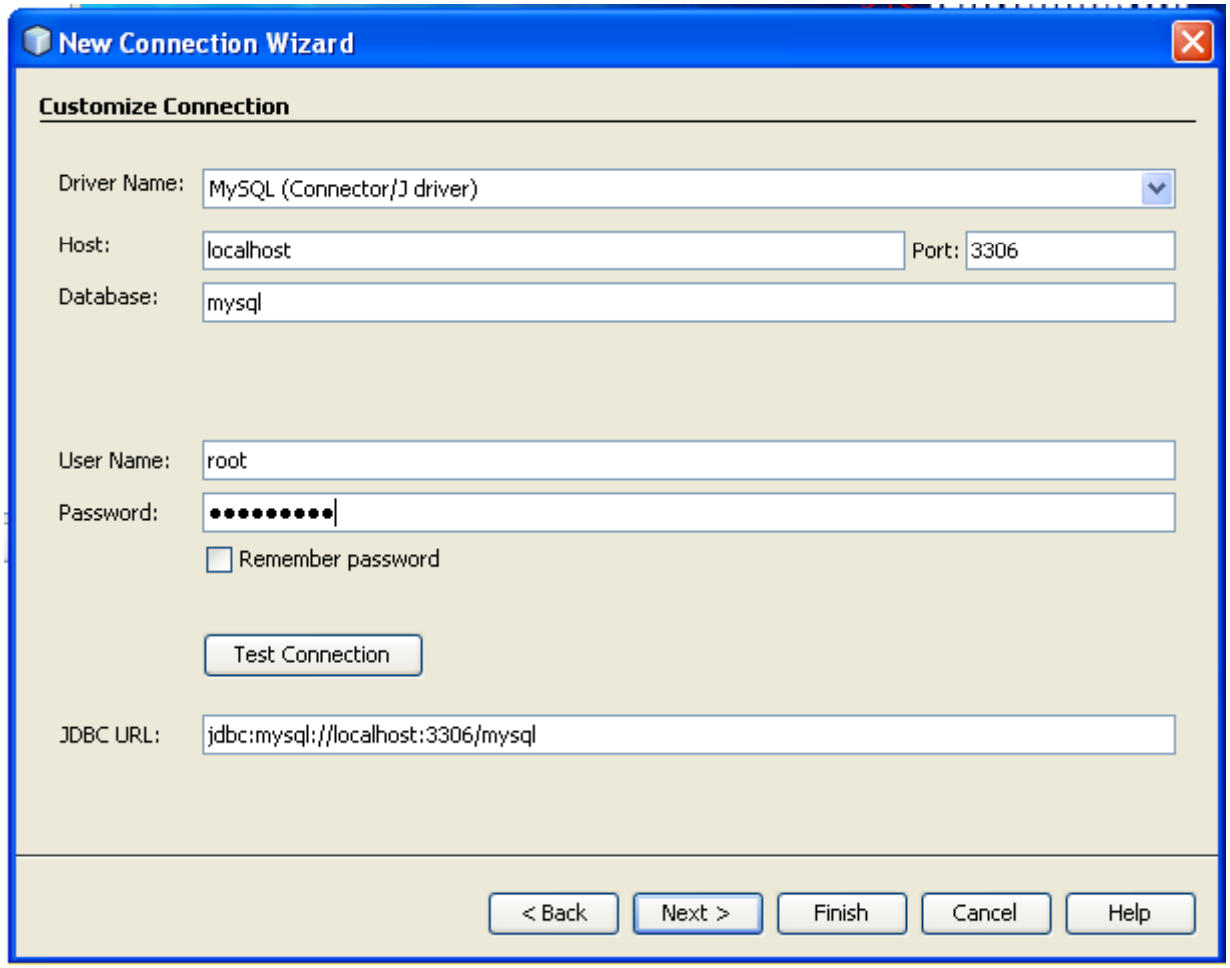


Connecting Database with Mysql Jconnector

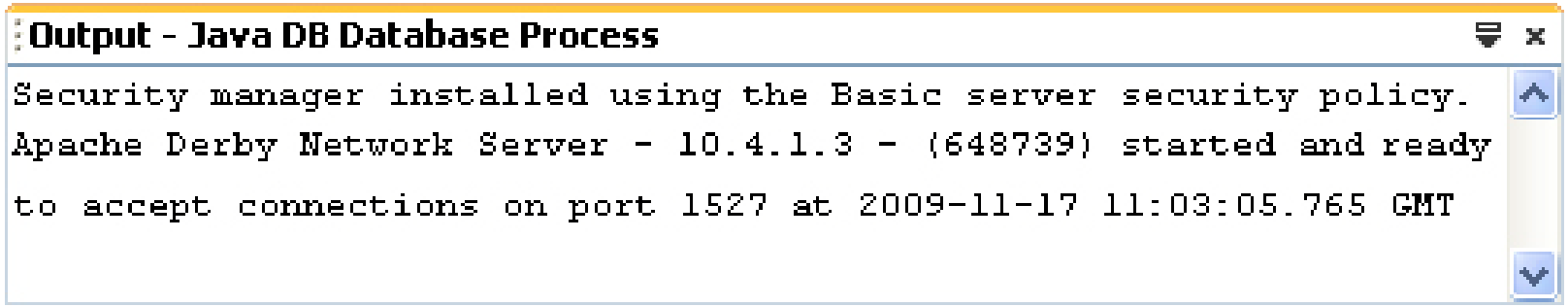
- The first thing to do, then, is to start the server.
- So right click on MySql (Connector/ j driver).
- You'll see a menu appear.
- Select **connect using:**



- Type password, then press test connection

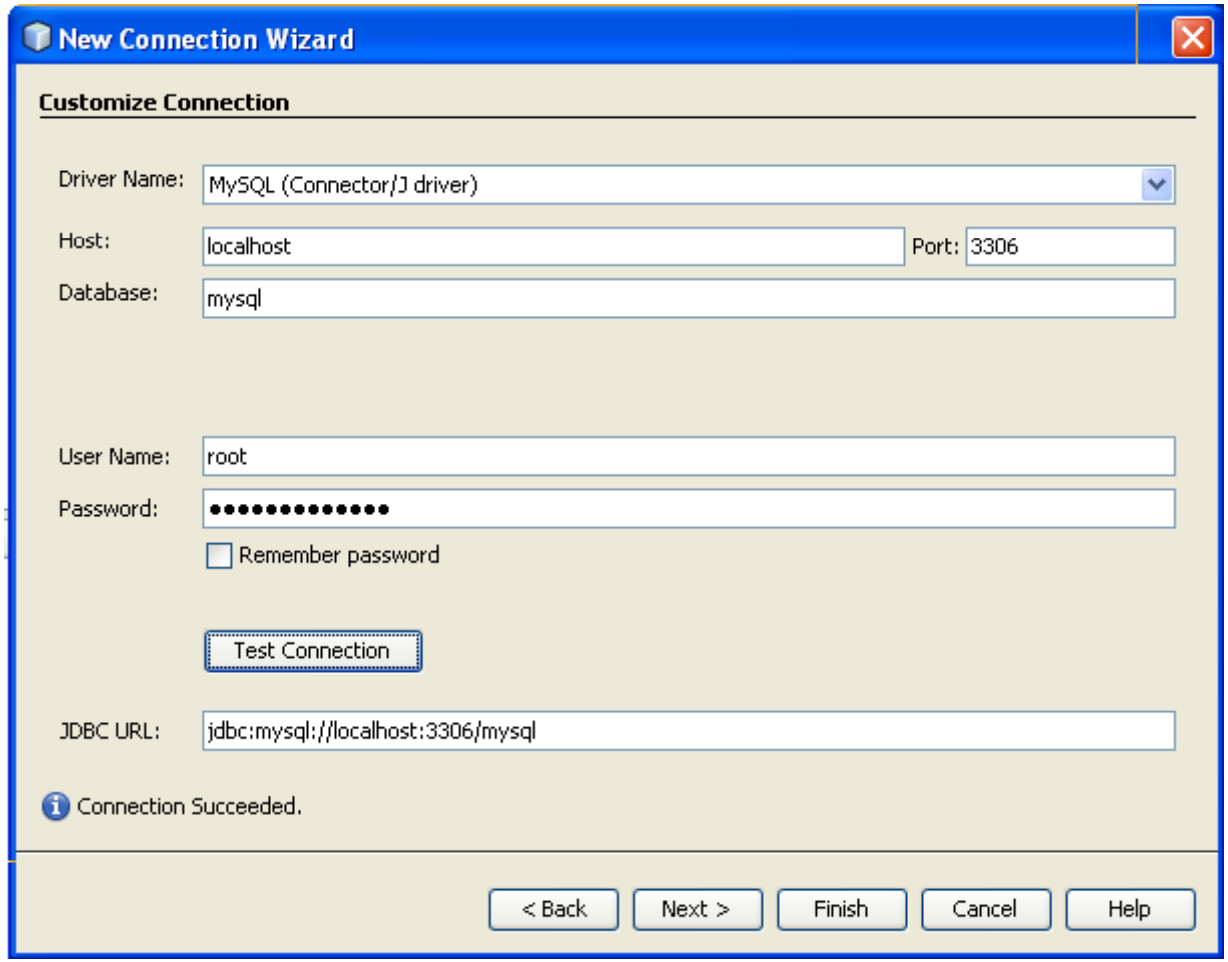


- Have a look at the Output window and you'll see a few messages appear: (If you have a firewall running, you'll need to let the MySql through.)

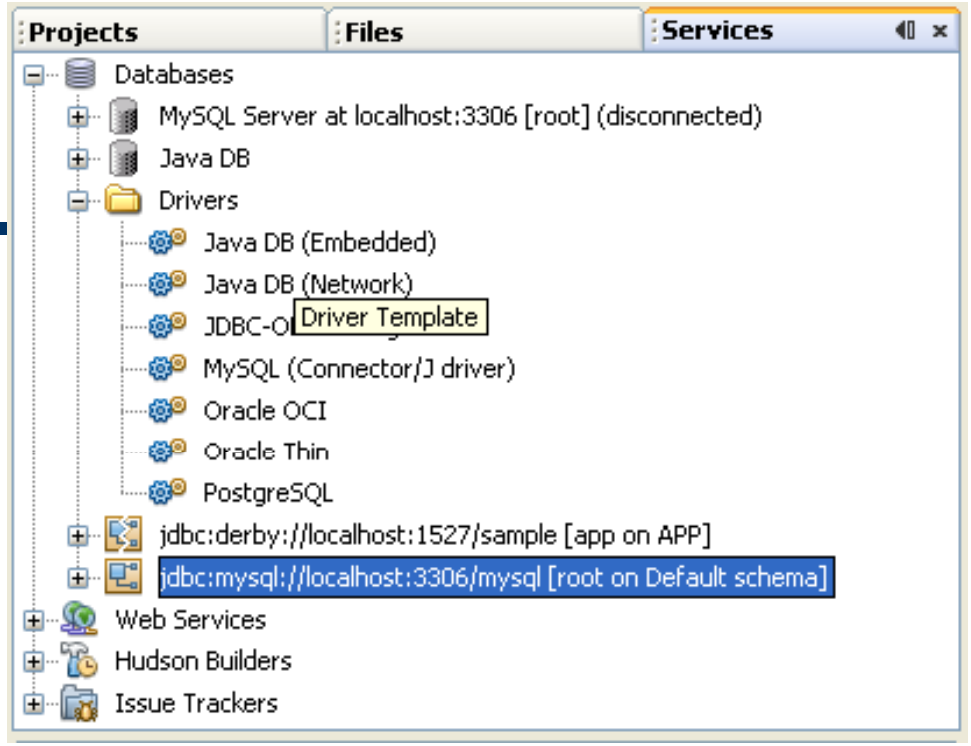


- If connection is ok, the following message will appear in the bottom of the form

- Press finish

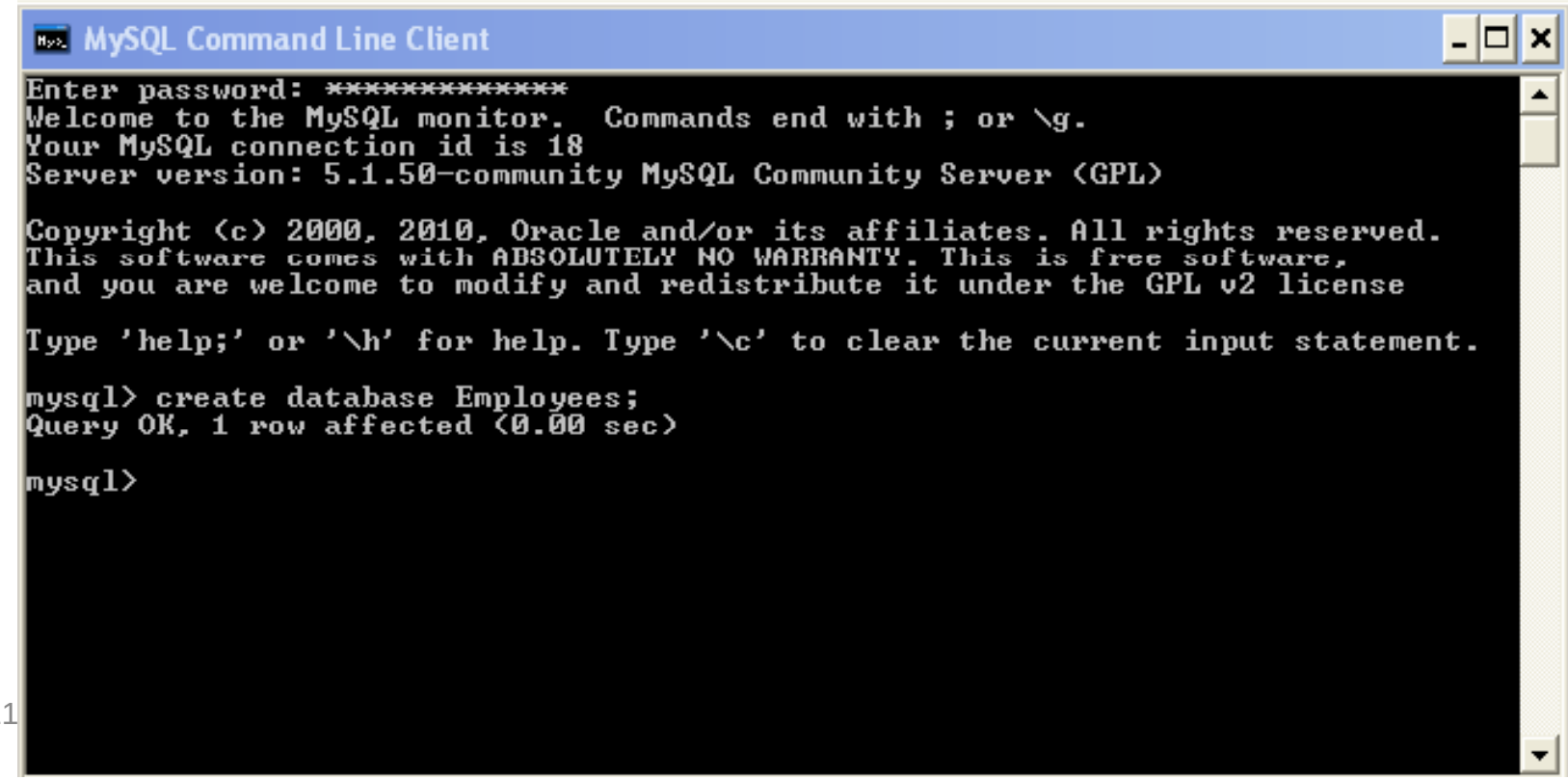


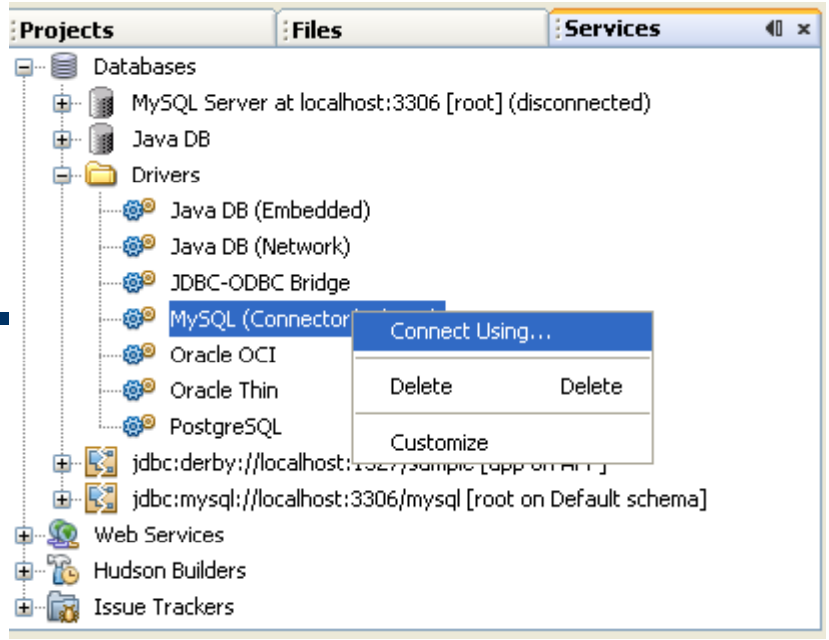
- The connection will appear in the DB tree
- Now you can connect and disconnect to DB by:
- Right click on the connection, select **connect** or **disconnect**
- If you choosed not to remember the password, a dialog box will appear to enter password



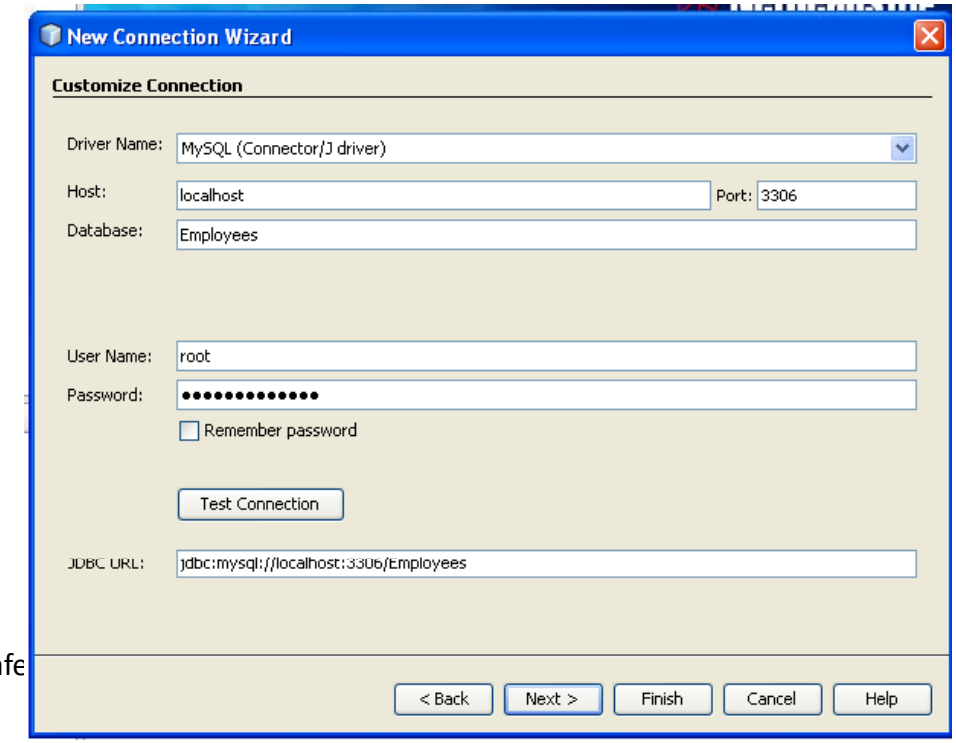
Create new DB with MySql (connector/J driver)

- Disconnect from mysql command line client
- Start MySql, type password
- Create new database;

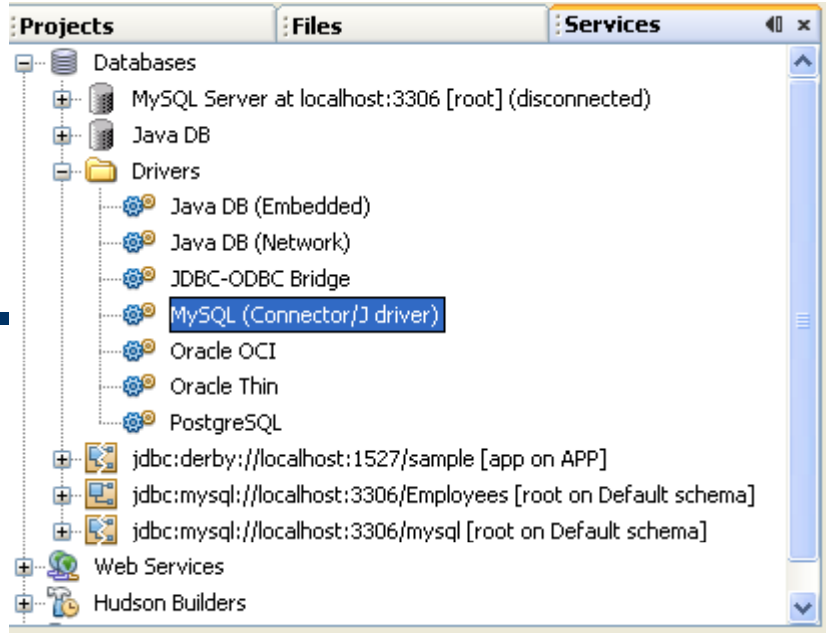


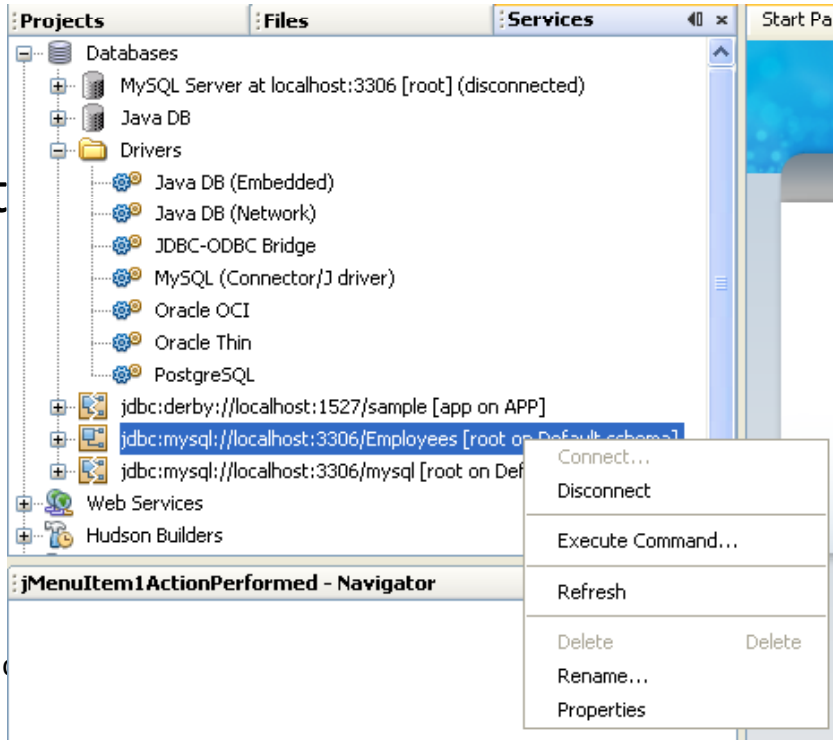


- Return to NetBeans
- Create a new connection to Emplyees db.
- Change Database name to **Emplyees**
- Type password, press test connection.
- If every thing is ok,
- Check remember password
- press finish.

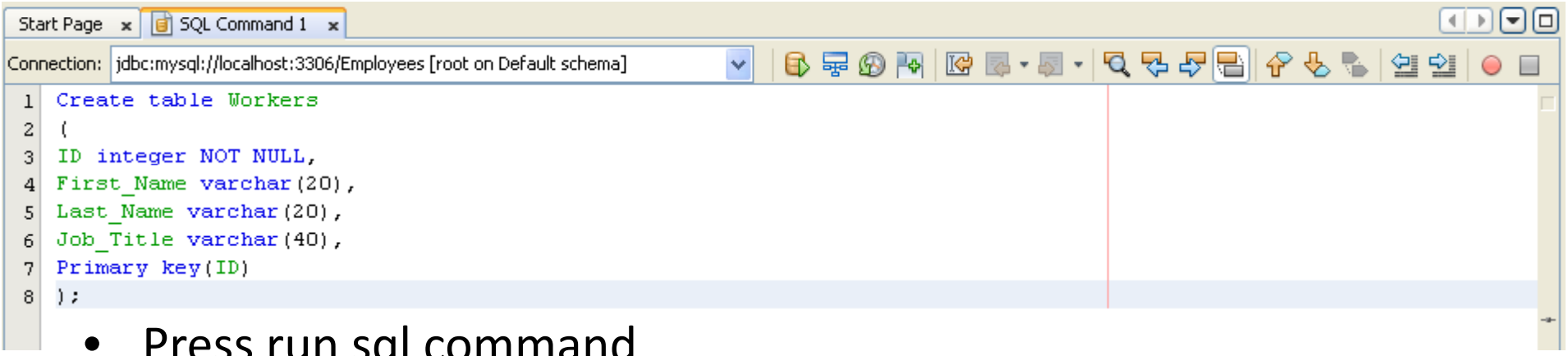


- Click OK to create your database. It should then appear on the list:
- you need to create a table in the database.
- To do so, right click on your database.
- From the menu that appears select **Execute command:**

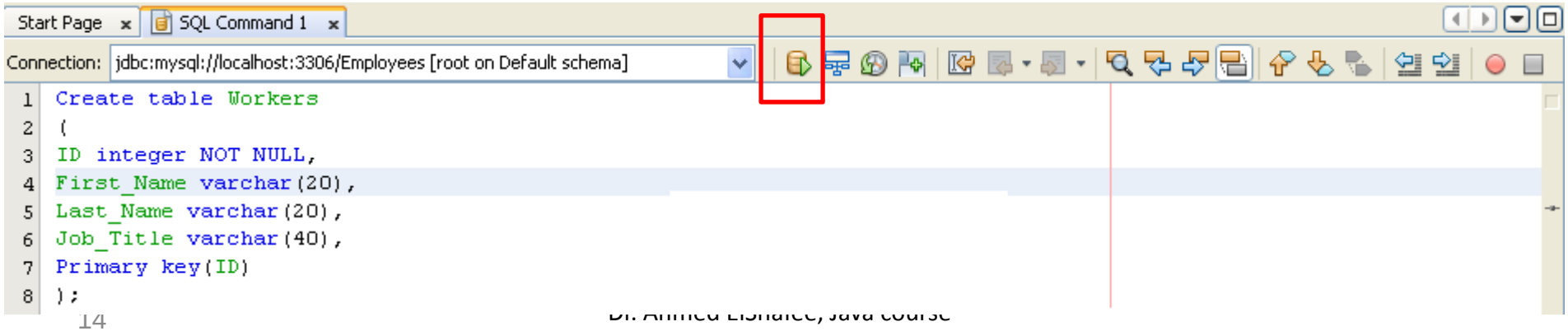




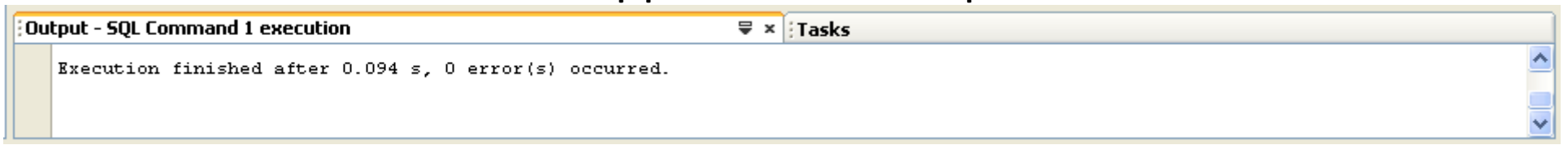
- A command windows will appear.
- Type the following



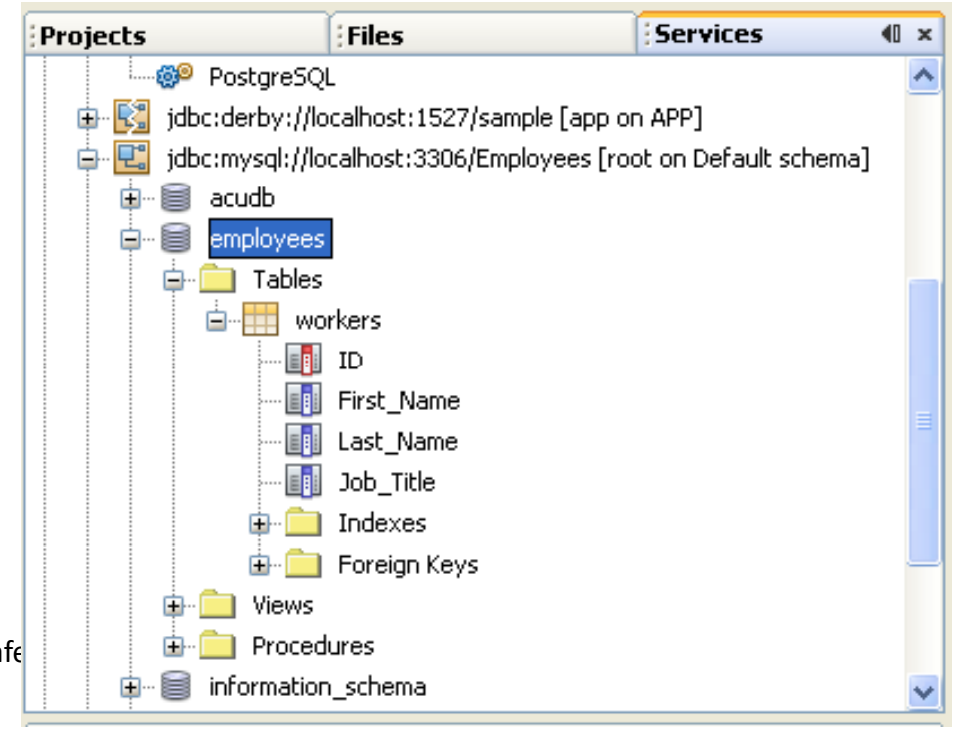
- Press run sql command



- Execution result will appear in the Output window

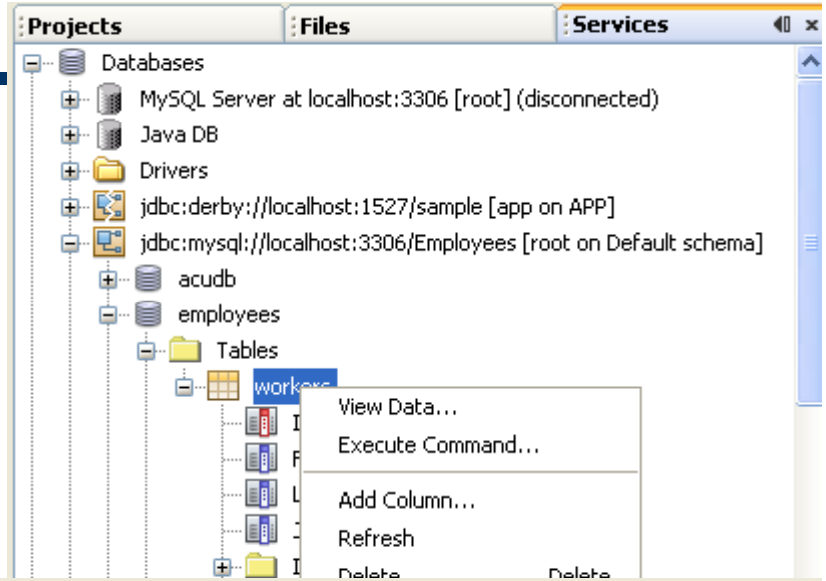


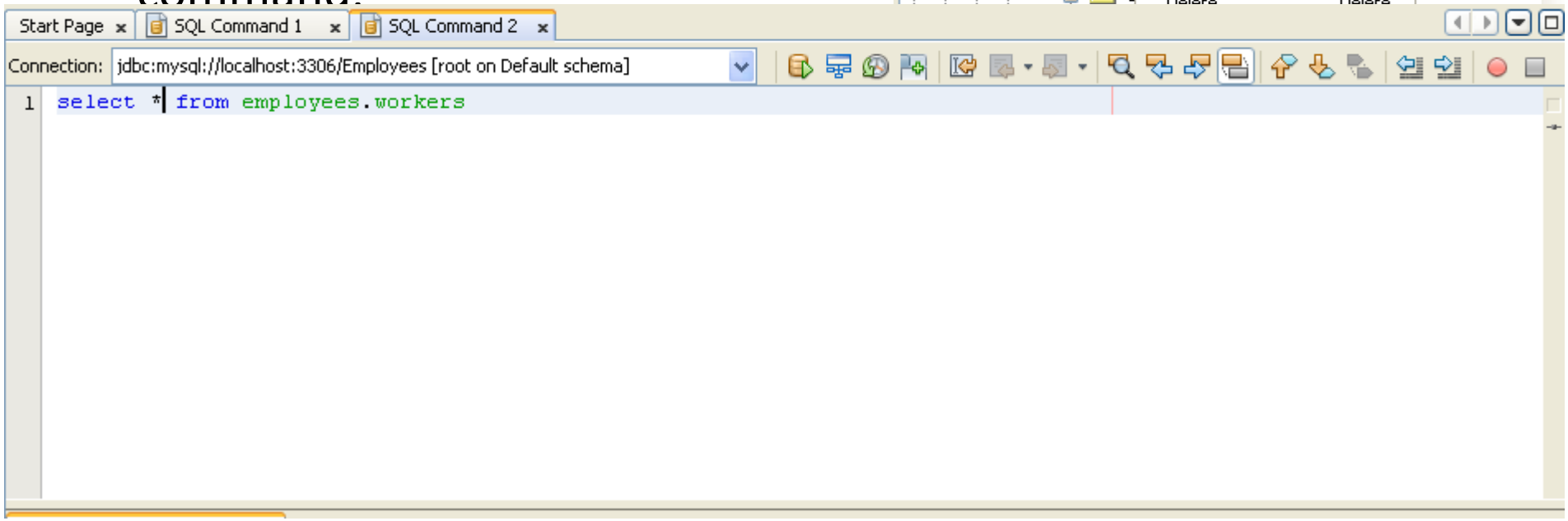
- Table and its columns will appear in the services window



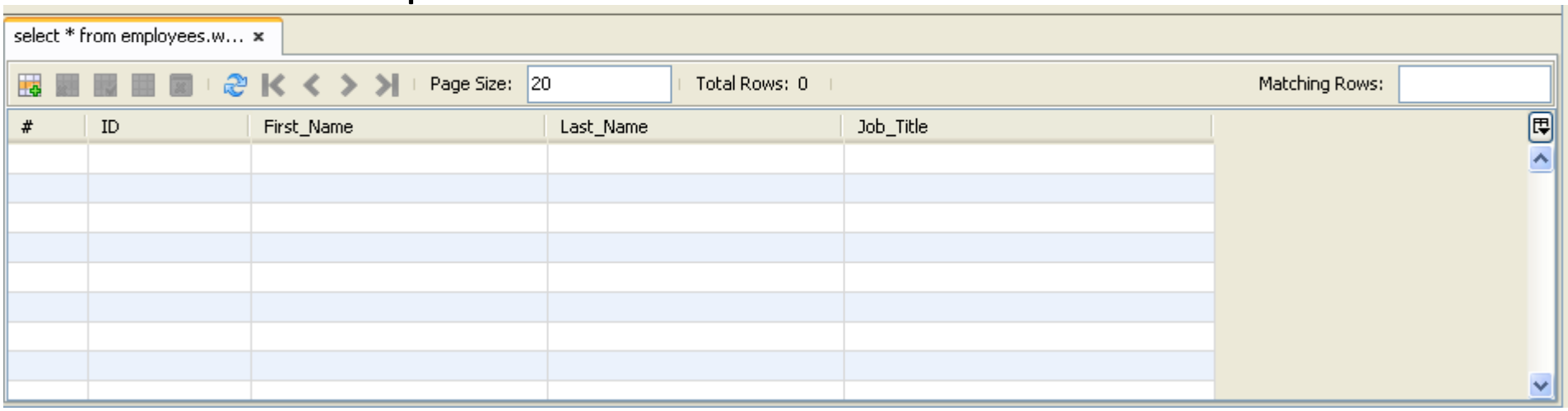
Adding Records to the Database Table

- To display data stored in db table,
- Right click, select **view data**
- A sql command window will appear contain **select** sql command.

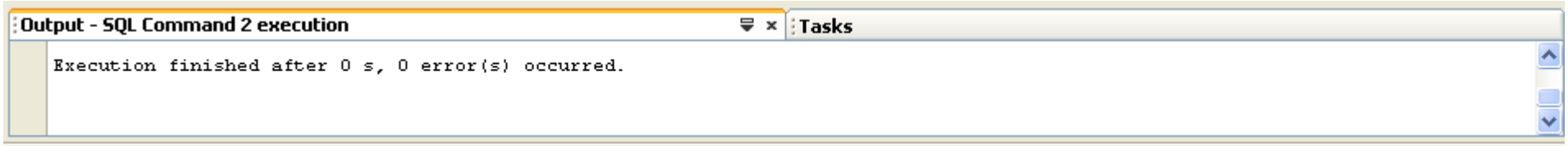




- You use the bottom half of the window to enter new table rows. The top half is for SQL Commands.

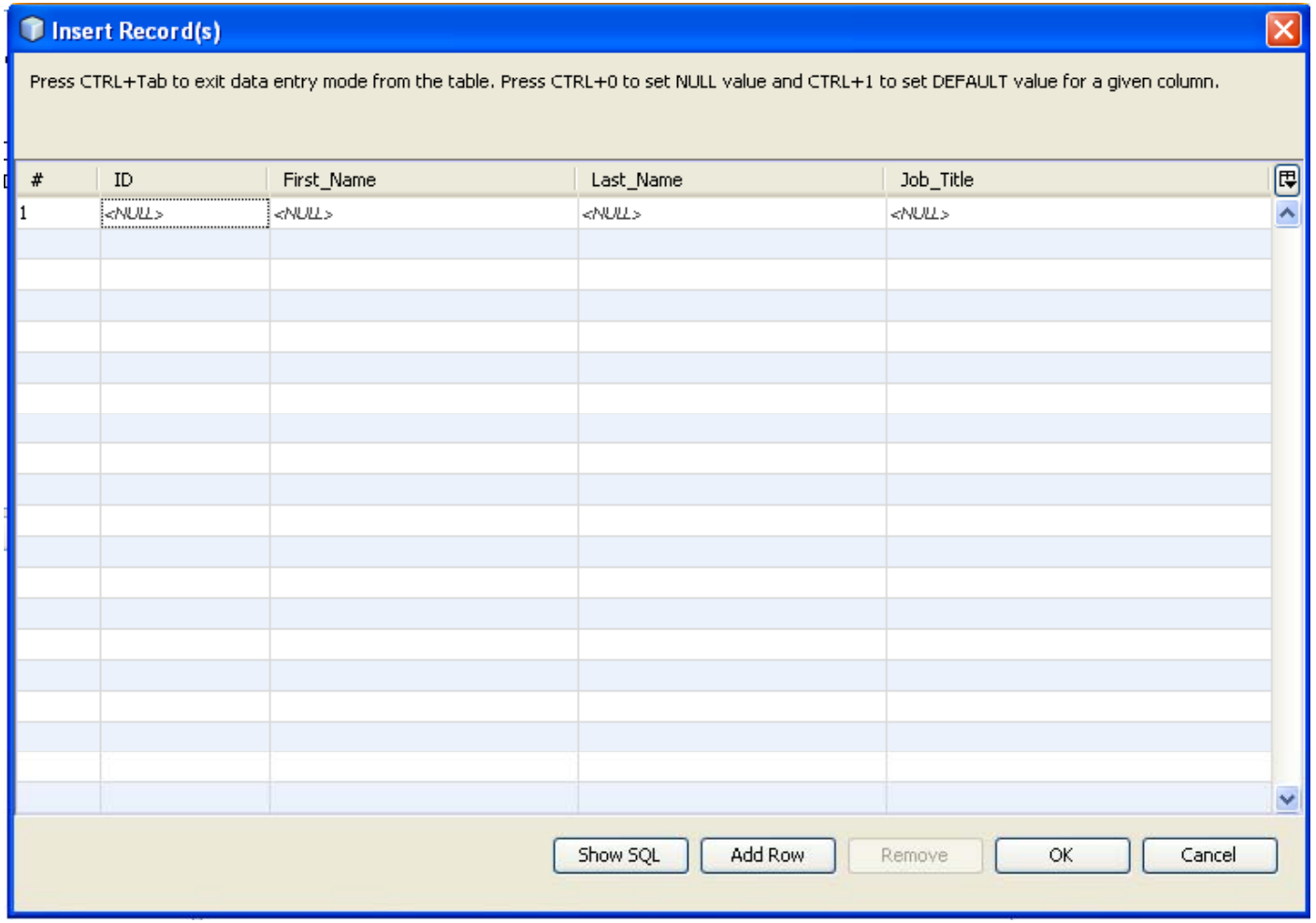


The output window contains sql command execution result.



- To add a new row, click the icon with the green plus symbol, in the bottom half of the window:

- When your click the new row icon, a dialogue box appears:



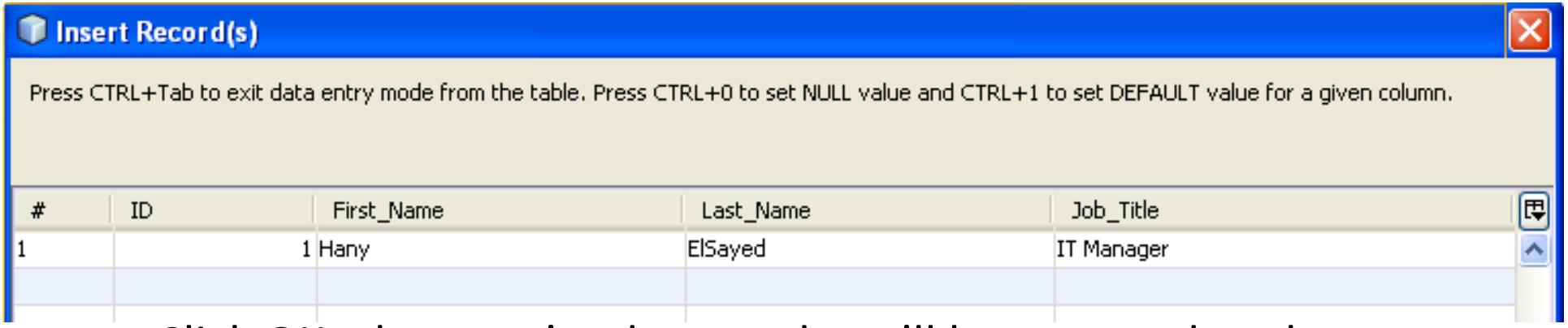
- Enter the following data as the first row of your table:

ID: 1

First Name: Hany

Last Name: ElSayed

Job Title: IT Manager



- Click OK when you're done and you'll be returned to the NetBeans window.
- The first row should then be displayed:

select * from employees.w... ×

Page Size: 20 Total Rows: 1 Page: 1 of 1 Matching Rows:

#	ID	First_Name	Last_Name	Job_Title
1	1	Hany	ElSayed	IT Manager

- Add 3 more records as shows below

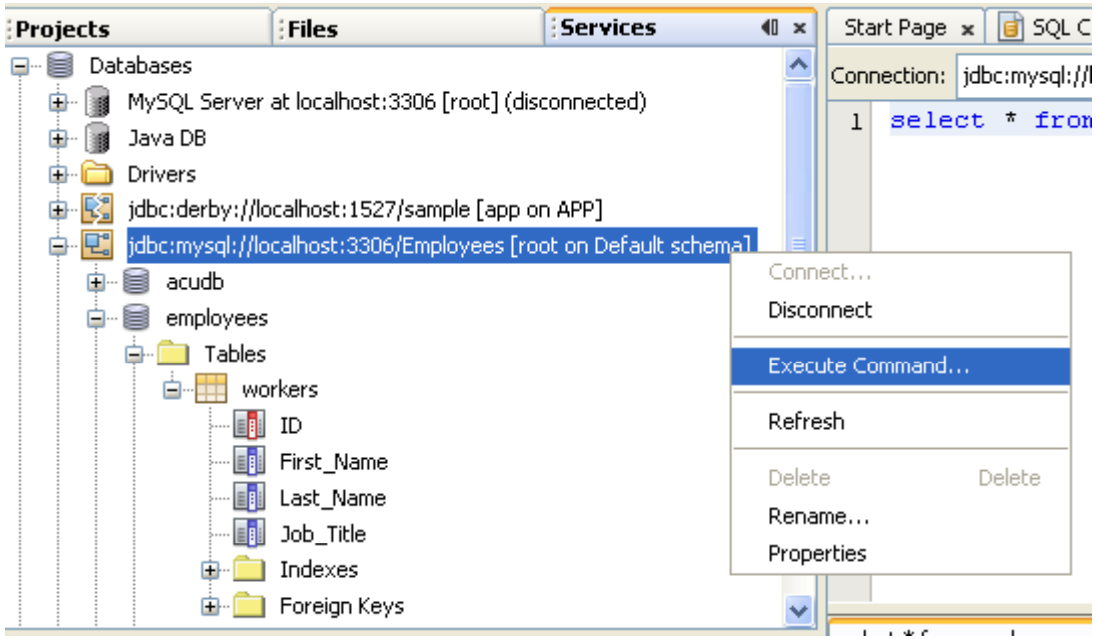
select * from employees.w... ×

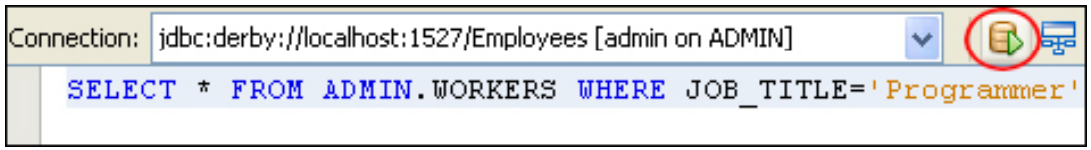
Page Size: 20 Total Rows: 4 Page: 1 of 1 Matching Rows:

#	ID	First_Name	Last_Name	Job_Title
1	1	Hany	ElSayed	IT Manager
2	2	Mostafa	Hussein	System Analyst
3	3	Mohamed	Gad	Programmer
4	4	Ibrahim	Gamal	Programmer

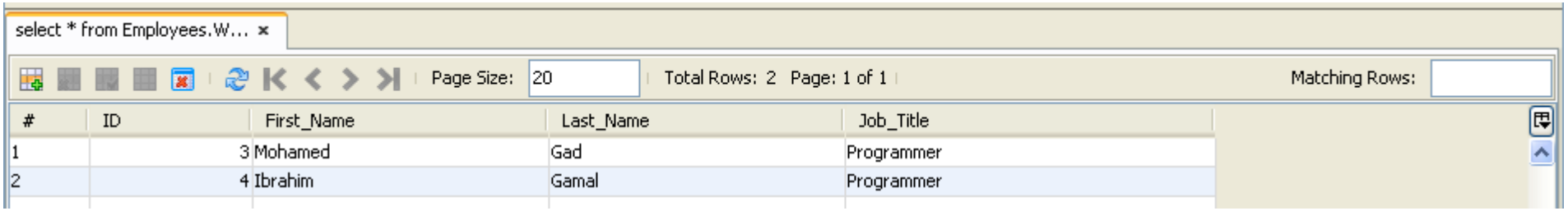
SQL Commands

- To try SQL Statement out, right-click your table name in the Services area.
- From the menu that appears, select **Execute Command:**
- When you click on **Execute Command**, a new window appears. Type the above SQL Statement, and then click the Run icon:

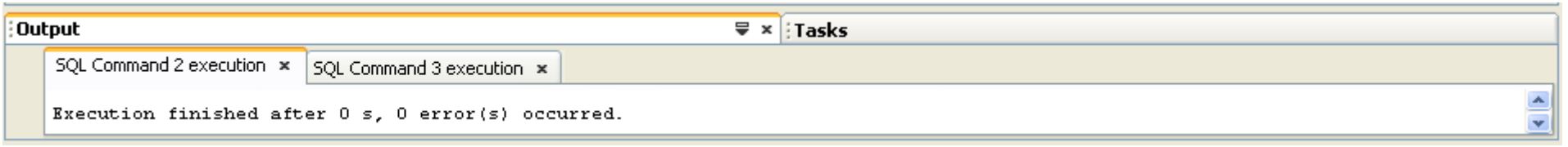




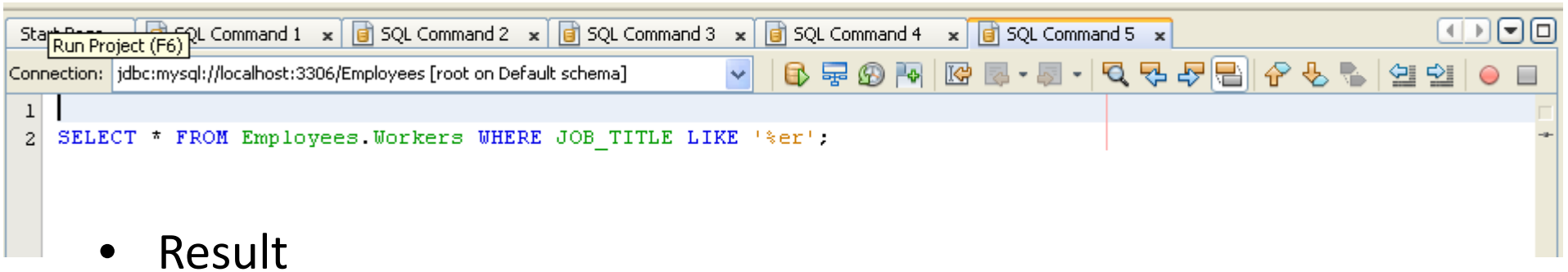
- The results will be displayed in the bottom half of the window:



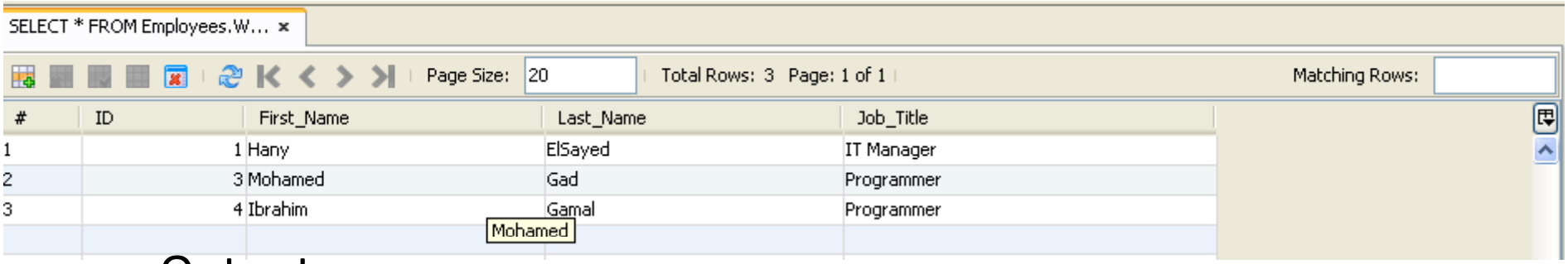
- The output window



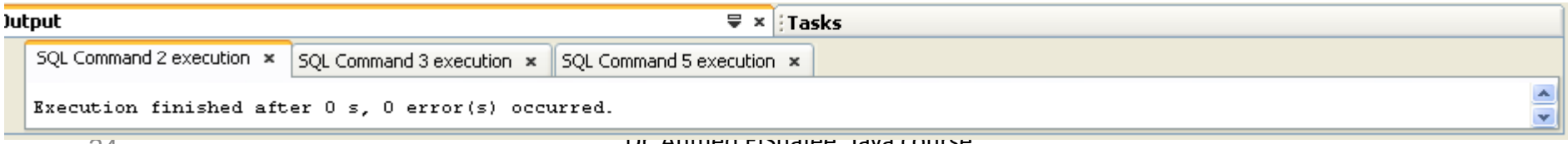
- Try this command too



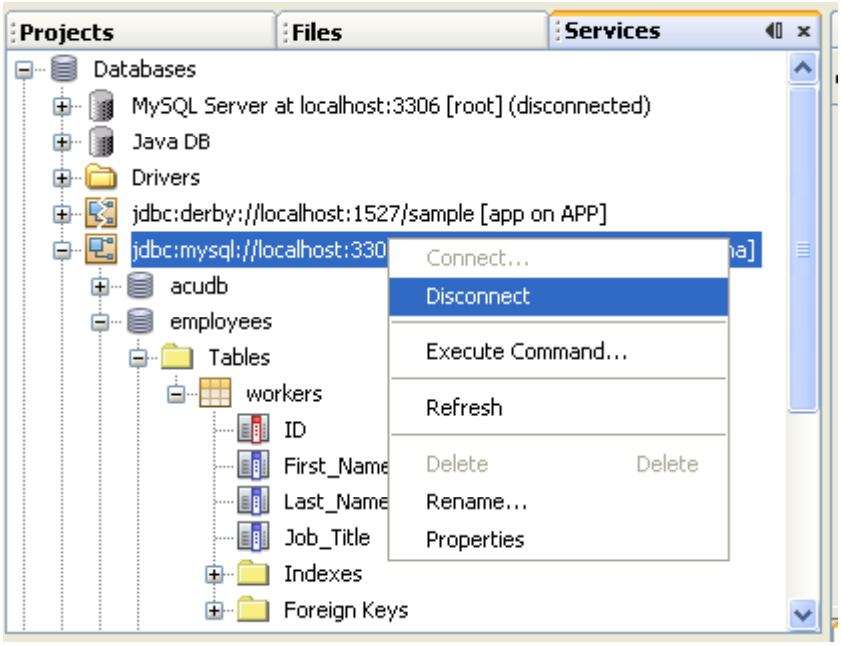
- Result



- Output

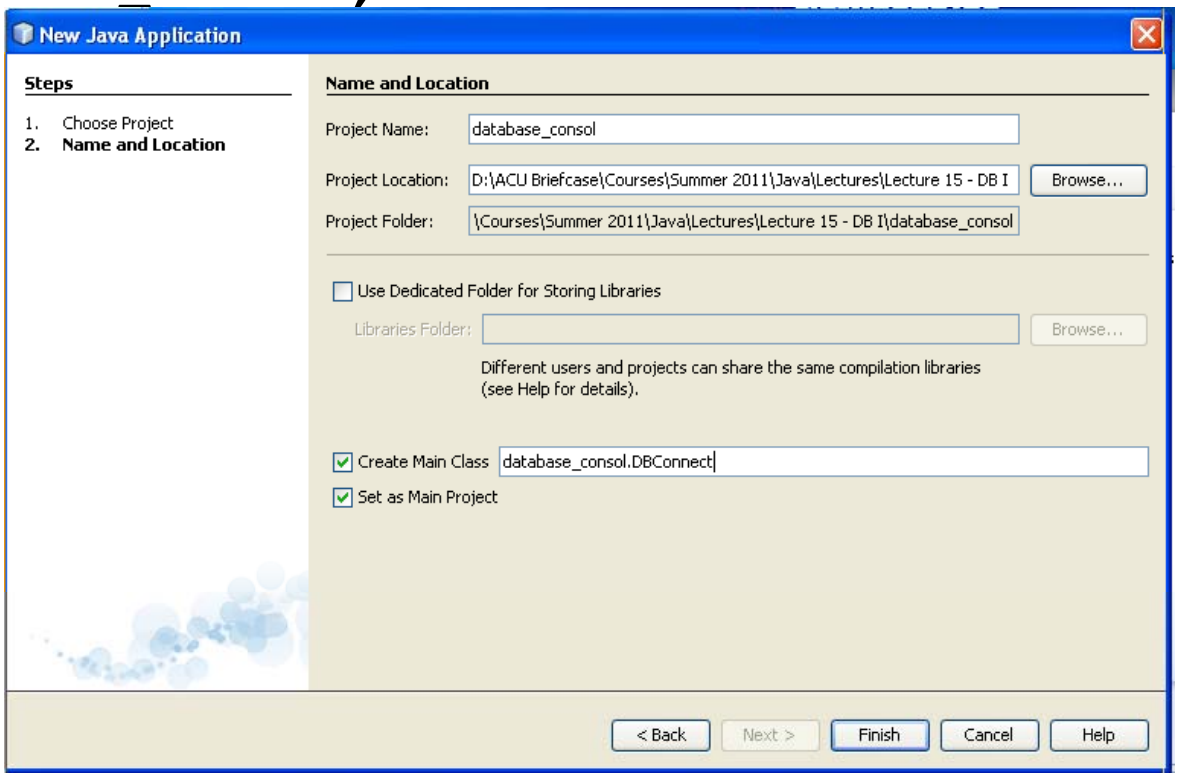


- We'll leave SQL Statements there, as we now have enough to start programming.
- Before doing so, however, close down the connection to your table by right clicking it in the Services are.
- From the menu, select **Disconnect:**



Connecting to a Database using Java code

- start a new project for this by clicking **File > New Project** from the NetBeans menu.
- Create a **Java Application.**
- Call the package **database_console,** and the Main class **DBConnect**:

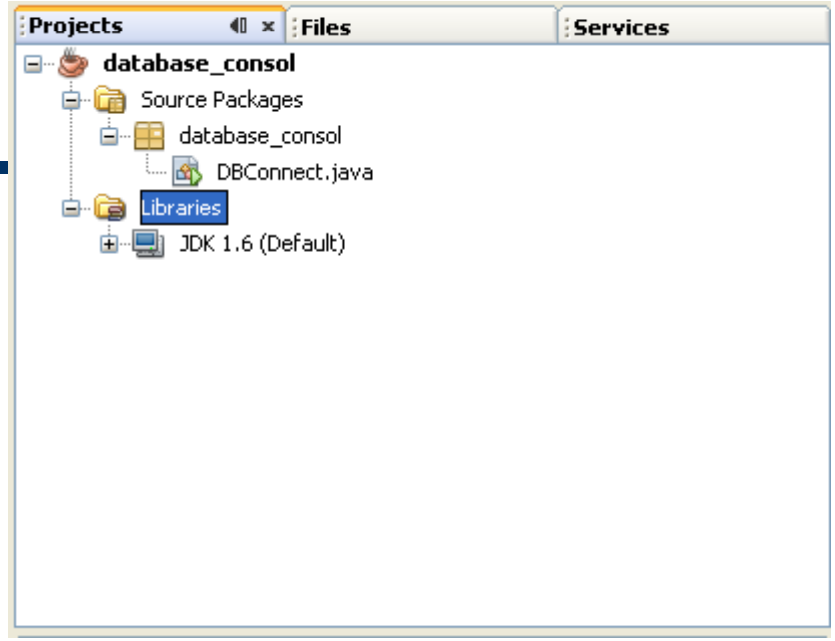


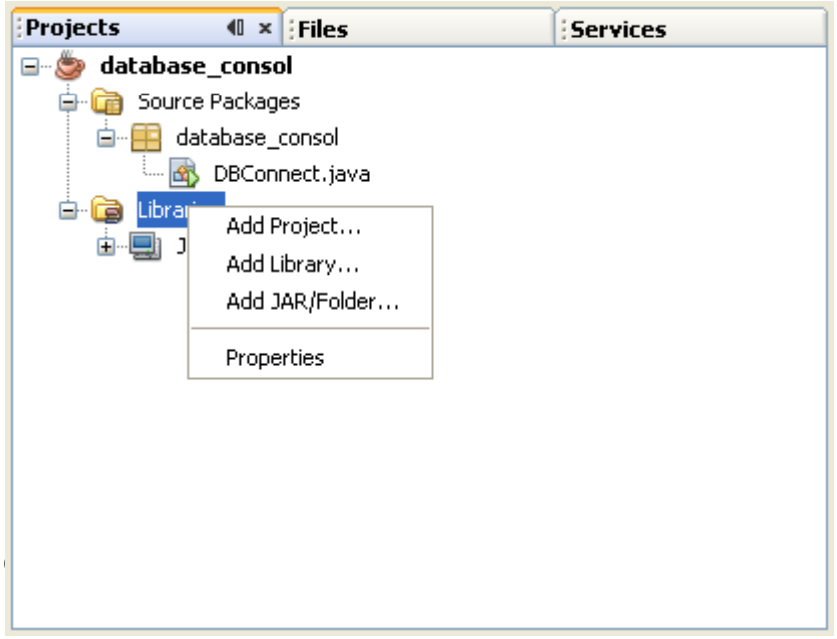
- When you click Finish, your code should look like this:

```
package database_console;

public class DBConnect {

    public static void main(String[] args) {

    }
}
```

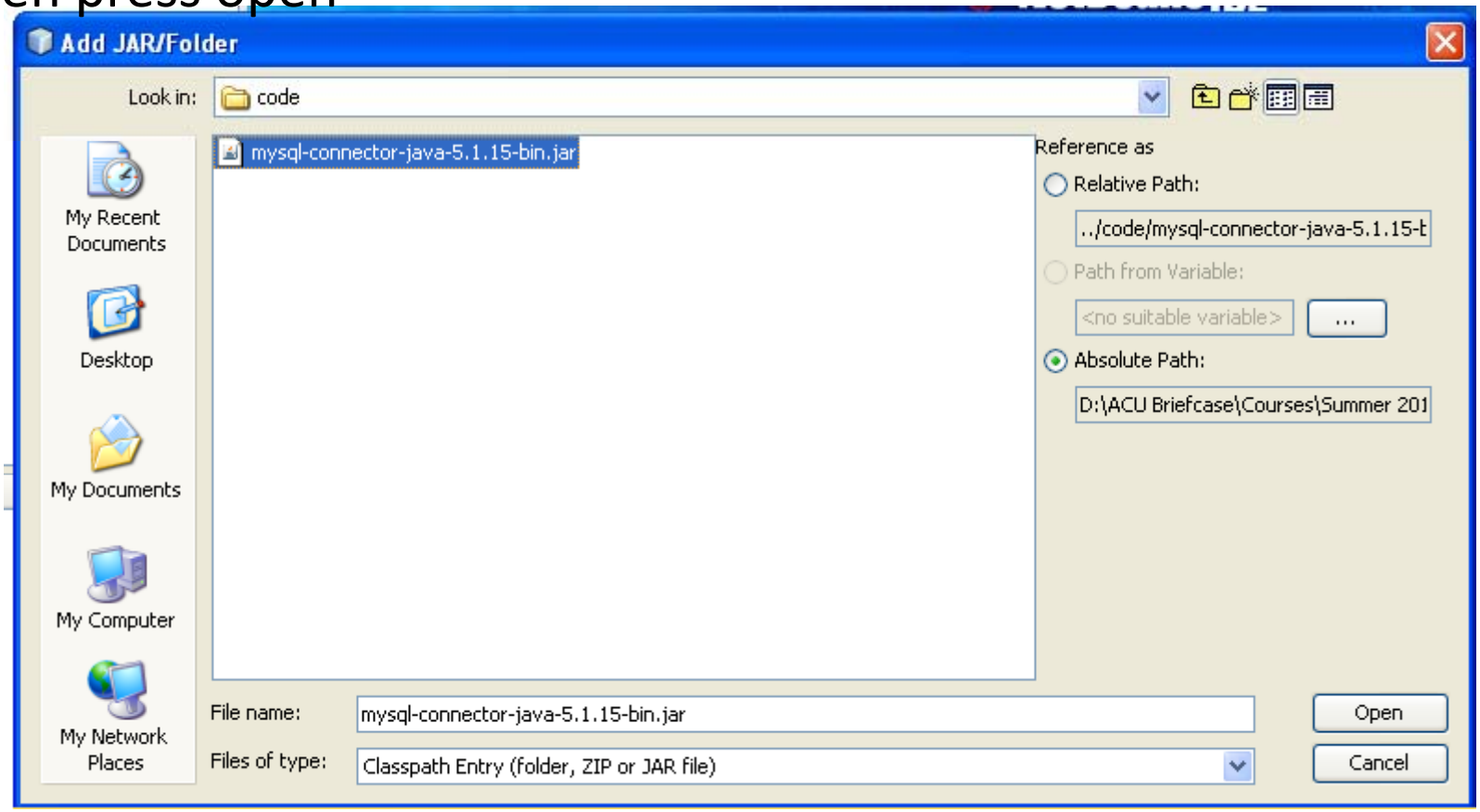
- To connect to a database you need a Connection object.
- The Connection object uses a DriverManager.
- The DriverManager passes in your database username, your password, and the location of the database.

- Now we need to add MySql j connector liberary.
- It’s a jar file contains necessary API to connect to MySql database server.
- In **projects** window, open **libraries**
- Right click on libraries and select add jar/folder

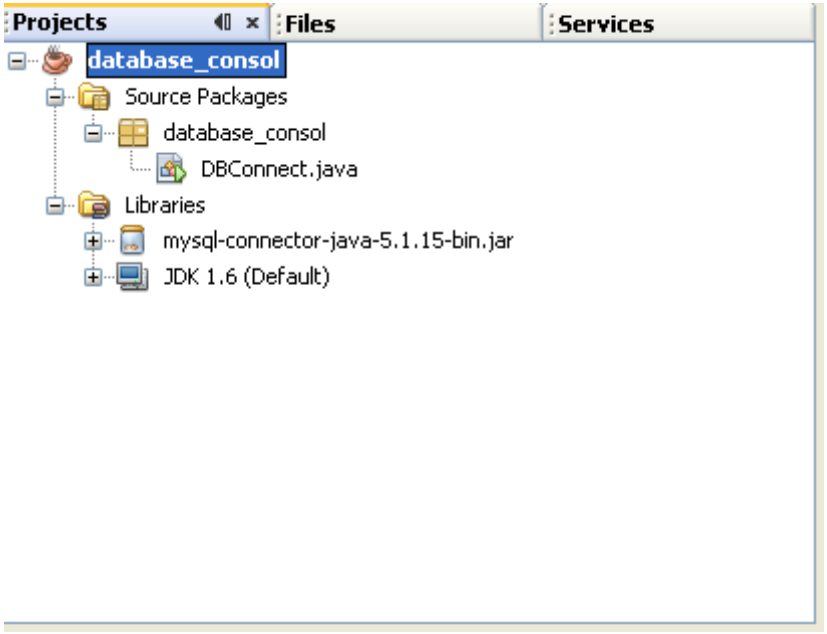




- Open dialog will appear, select MySql connector “mysql-connector-java-5.1.15-bin.jar”.
- hen press open



- MySql connector will appear in libraries tree



- Add these three import statements to the top of your code:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

- To set up a connection to a database, the code is this:

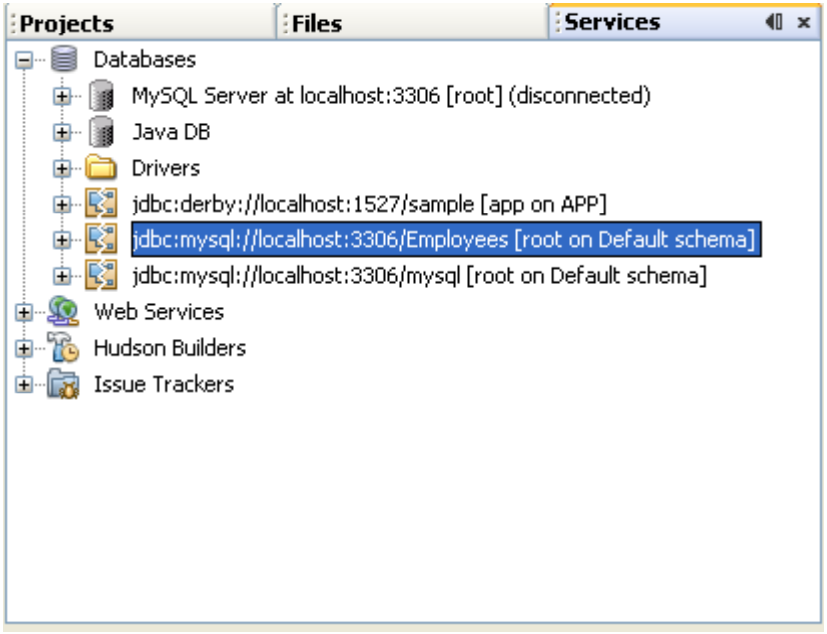
Connection con = DriverManager.getConnection(host, username, password);

- DriverManager has a method called **getConnection.**
- This needs a **host name** (which is the location of your database), a **username**, and a **password**.
- If a connection is successful, a Connection object is created, which we've called **con.**

- You can get the host address by looking at the Services tab on the left of NetBeans:
- The address of the highlighted database above is:

jdbc:mysql://localhost:3306/Employees“

- The first part **jdbc:mysql://localhost:3306** is the database type and server that you’re using.



- The 3306 is the port number.
- The database is Employees.
- This can all go in a String variable:
- so the string host will be

String url = "jdbc:mysql://localhost:3306/Employees";

- Two more strings can be added for the username and password:

String uName = "Your_Username_Here";

String uPass= " Your_Password_Here ";

- The code,

DBConnect.java

```
1  /*...*/
5  package database_consol;
6  import java.sql.Connection;
7  import java.sql.DriverManager;
   import java.sql.SQLException;
9
10 /**...*/
14 public class DBConnect {
15
16     /**...*/
19     public static void main(String[] args) {
20         // TODO code application logic here
21                 String uName = "root";
22                 String uPass = "root";
23                 String host = "jdbc:mysql://localhost:3306/Employees";
                   Connection con = DriverManager.getConnection( host, uName, uPass );
25
26
27     }
28 }
29
```

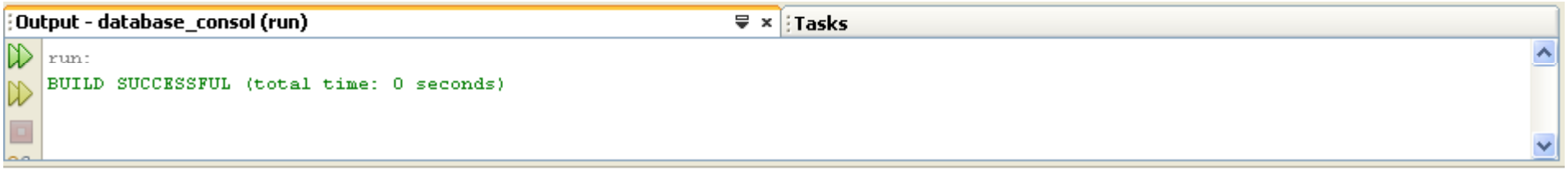
- As you can see in the image above, there is a wavy underline for the Connection code.
- The reason for this is because we haven't trapped a specific error that will be thrown up when connecting to a database – the **SQLException error.**
- It's the DriverManager that attempts to connect to the database.
- If it fails (incorrect host address, for example) then it will hand you back a **SQLException** error.
- You need to write code to deal with this potential error.
- In the code below, we're trapping the error in **catch part of the try … catch statement:**

```
try {
}
catch ( SQLException err ) {
System.out.println( err.getMessage( ) );
}
```

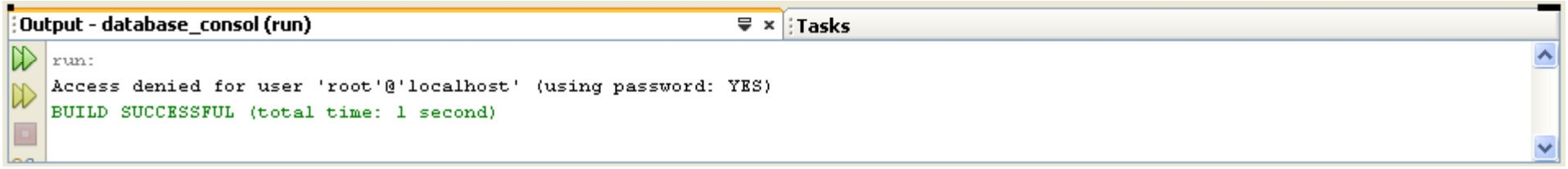
- In between the round brackets of catch, we've set up a SQLException object called **err.**
- We can then use the **getMessage** method of this **err** object**.**
- Add the above **try …catch** block to your own code, and move your four connection lines of code to the **try** part**.**
- Your code will then look like this:

```
/**...*/
public class DBConnect {

    /**...*/
    public static void main(String[] args) {
        // TODO code application logic here
        try{
                String uName = "root";
                String uPass = "pass";
                String host = "jdbc:mysql://localhost:3306/Employees";
                Connection con = DriverManager.getConnection( host, uName, uPass );
        }
        catch(SQLException err){
        System.out.println(err.getMessage());
        }

    }
}
```

- Run, if every thing is fine you will got nothing



- If error occurred you will got error message



Connecting to a Database Table

- Now that you have connected to the database, the next step is to access the table in your database.
- For this, you need to execute a SQL Statement, and then manipulate all the rows and columns that were returned.
- To execute a SQL statement on your table, you set up a Statement object.
- So add this import line to the top of your code:

import java.sql.Statement;

- In the **try** part of the try **… catch** block add the following line **(add it just below** your Connection line):

Statement stmt = con.createStatement( );

- Here, we're creating a Statement object called **stmt.**
- The **Statement object** needs a **Connection object**, with the **createStatment** method.
- We also need a SQL Statement for the Statement object to execute.
- So add this line to your code:

String SQL = "SELECT * FROM Workers";

- The above statement selects all the records from the database table called Workers.

- We can pass this SQL query to a method of the Statement object called **executeQuery.**
- The Statement object will then go to work gathering all the records that match our query.
- However, the **executeQuery** method returns all the records in something called a **ResultSet**.
- Before we explain what these are, add the following import line to the top of your code:

import java.sql.ResultSet;

- Now add this line just below your SQL String line:

ResultSet rs = stmt.executeQuery(SQL);

- So our ResultSet object is called **rs.**
- This will hold all the records from the database table.
- Before we go any further, though, here's an explanation of what **ResultSets** are.

- Your code now,…

```
package database_consol;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

/**...*/
public class DBConnect {

    /**...*/
    public static void main(String[] args) {
        // TODO code application logic here
        try{
                String uName = "root";
                String uPass = "root";
                String host = "jdbc:mysql://localhost:3306/Employees";
                Connection con = DriverManager.getConnection( host, uName, uPass );
                Statement stmt = con.createStatement( );
                String SQL = "SELECT * FROM Workers";
                ResultSet rs = stmt.executeQuery( SQL );
        }
        catch(SQLException err){
```

ResultSets

- A **ResultSet** is a way to store and manipulate the records returned from a SQL query.
- **ResultSets** come in three different types.
- The type you use depends on what you want to do with the data:
- 1) Do you just want to move forward through the records, from beginning to end?
- 2) Do you want to move forward AND backward through the records, as well as detecting any changes made to the records?

(3) Do you want to move forward AND backward through the records, but are not bothered about any changes made to the records

- Type number 1 on the list above is called a **TYPE_FORWARD_ONLY ResultSet**.
- Number 2 on the list is a **TYPE_SCROLL_SENSITIVE ResultSet**.
- The third ResultSet option is called **TYPE_SCROLL_INSENSITIVE.**
- The ResultSet type goes between the round brackets of createStement:

Statement stmt = con.createStatement( );

- Because we've left the round brackets empty, we'll get the default RecordSet, which is **TYPE_FORWARD_ONLY**.
- we'll use one of the other types.
- But you use them like this:

Statement stmt =

con.createStatement(RecordSet.TYPE_SCROLL_SENSITIVE);

- So you first type the word RecordSet. After a dot, you add the RecordSet type you want to use.

- you also need to specify whether the **ResultSet** is Read Only or whether it is Updatable.
- You do this with two built-in constants: **CONCUR_READ_ONLY and CONCUR_UPDATABLE**.
- Again, these come after the word RecordSet:

ResultSet.CONCUR_READ_ONLY

ResultSet.CONCUR_UPDATABLE

- The final code will be,…

```
Statement stmt =con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDA
TABLE);
```

- One more thing to get used to with **ResultSets** is something called a **Cursor**.
- A **Cursor** is really just a pointer to a table row.
- When you first load the records into a **ResultSet**, the **Cursor** is pointing to just before the first row in the table.
- You then use methods to manipulate the **Cursor**.
- But the idea is to identify a particular row in your table.

Using a ResultSet

- Once you have all the records in a Results set, there are methods you can use to manipulate your records.
- Here are the methods you'll use most often:

Next	Moves the Cursor to the next row in your table. If there are no more rows in the table, a value of False will be returned.
Previous	Moves the Cursor back one row in your table. If there are no more rows in the table, a value of False will be returned.

first	Moves the Cursor to the first row in your table
last	Moves the Cursor to the last row in your table
Absolute	Moves the Cursor to a particular row in the table. So absolute(5) will move the Cursor to row number 5 in the table

- The ResultSet also has methods you can use to identify a particular column (field) in a row.
- You can do so either by using the name of the column, or by using its index number.
- For our Workers table we set up four columns.
- They had the following names: **ID, First_Name, Last_Name, and Job_Title**.
- The index numbers are therefore 1, 2, 3, 4.

- We set up the ID column to hold Integer values. The method you use to get at integer values in a column is **getInt**:

int id_col = rs.getInt(“ID”);

- We could use the Index number instead:

int id_col = rs.getInt(1);

- For the other three columns in our database table, we set them up to hold Strings.
- We, therefore, need the getString method:

String first_name = rs.getString(“First_Name”);

- Or we could use the Index number:

String first_name = rs.getString(2);

- Because the ResultSet Cursor is pointing to just before the first record when the data is loaded, we need to use the **next** method to move to the first row.
- The following code will get the first record from the table:

```
rs.next( );
int id_col = rs.getInt("ID");
String first_name = rs.getString("First_Name");
String last_name = rs.getString("Last_Name");
String job = rs.getString("Job_Title");
```

- You can add a print line to your code

```
System.out.println( id_col + " " + first_name + " " + last_name +
                    " " + job );
```

- Code,..

```
public static void main(String[] args) {
    // TODO code application logic here
    try{
            String uName = "root";
            String uPass = "root";
            String host = "jdbc:mysql://localhost:3306/Employees";
            Connection con = DriverManager.getConnection( host, uName, uPass );
            Statement stmt =con.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
            String SQL = "SELECT * FROM Workers";
            ResultSet rs = stmt.executeQuery( SQL );
            rs.next( );
             int id_col = rs.getInt("ID");
             String first_name = rs.getString("First_Name");
             String last_name = rs.getString("Last_Name");
             String job = rs.getString("Job_Title");
             System.out.println( id_col + " " + first_name + " " + last_name + " " + job );
    }
    catch(SQLException err){
    System.out.println(err.getMessage());
    }

}
```

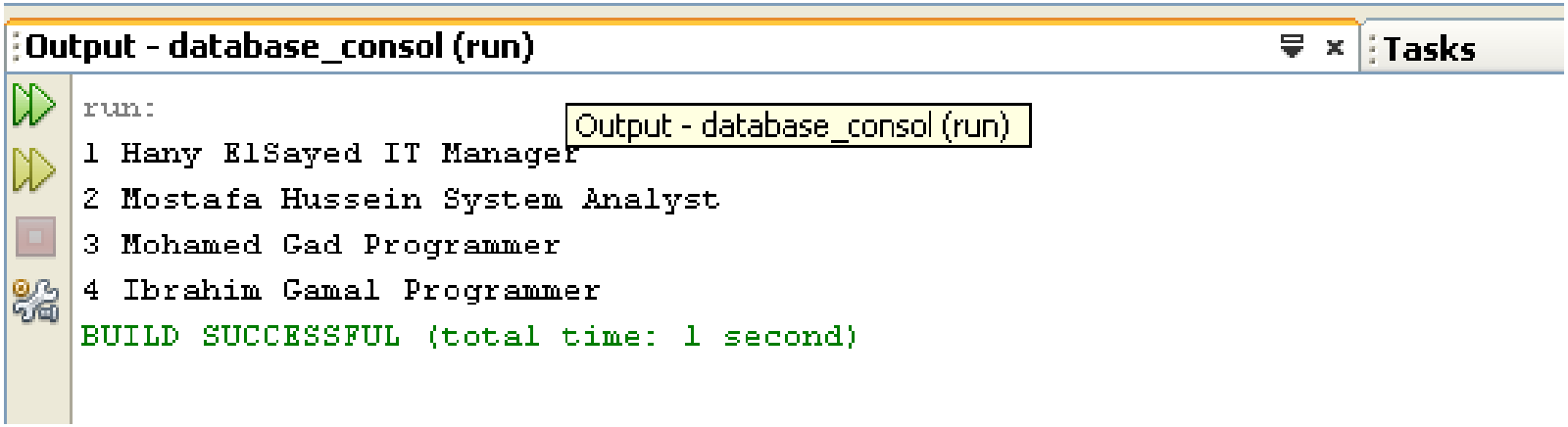
- If you want to go through all the records in the table, you can use a loop.
- Because the next method returns true or false, you can use it as the condition for a while loop:

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

Code,…

```
public static void main(String[] args) {
    // TODO code application logic here
    try{
            String uName = "root";
            String uPass = "root";
            String host = "jdbc:mysql://localhost:3306/Employees";
            Connection con = DriverManager.getConnection( host, uName, uPass );
            Statement stmt =con.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
            String SQL = "SELECT * FROM Workers";
            ResultSet rs = stmt.executeQuery( SQL );
            while(rs.next( ))
            {
             int id_col = rs.getInt("ID");
             String first_name = rs.getString("First_Name");
             String last_name = rs.getString("Last_Name");
             String job = rs.getString("Job_Title");
             System.out.println( id_col + " " + first_name + " " + last_name + " " + job );
            }
    }
    catch(SQLException err){
    System.out.println(err.getMessage());
    }
```

Run,…


Output - database_consol (run)
Tasks
run:
Output - database_consol (run)
1 Hany ElSayed IT Manager
2 Mostafa Hussein System Analyst
3 Mohamed Gad Programmer
4 Ibrahim Gamal Programmer
BUILD SUCCESSFUL (total time: 1 second)

Thanks,

See you next Lecture, isA