

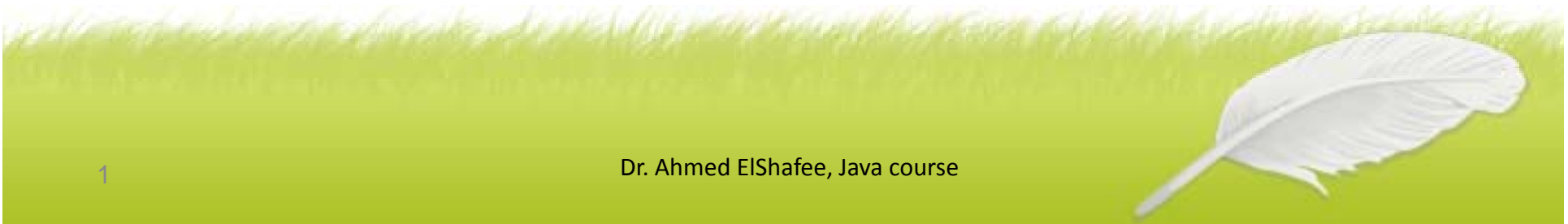


# Lecture (15)

## DB I

---

Dr. Ahmed ElShafee



# Agenda

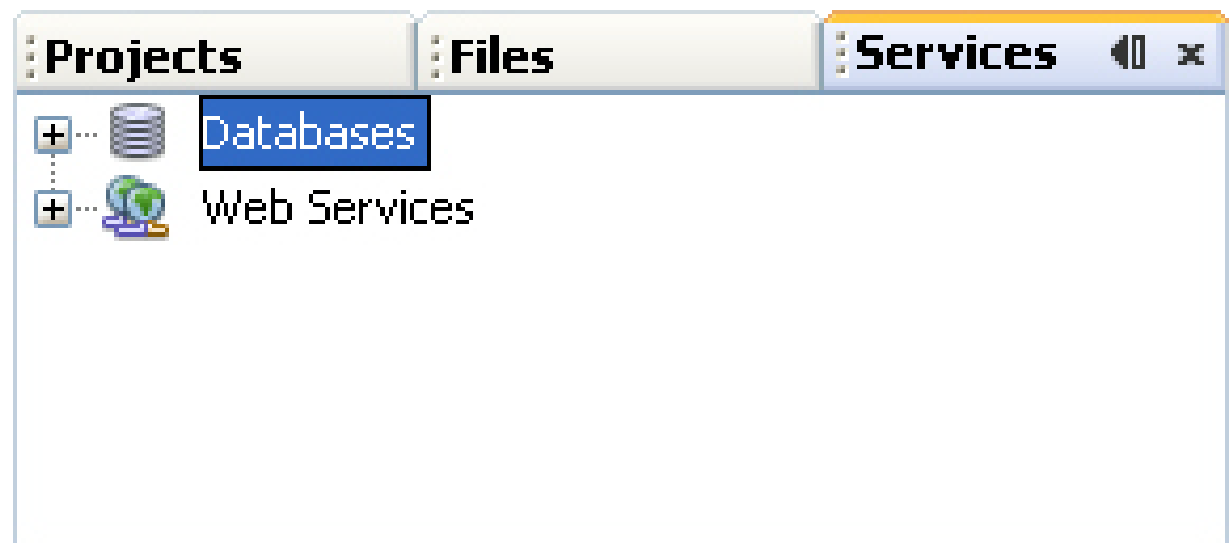
---

# Introduction

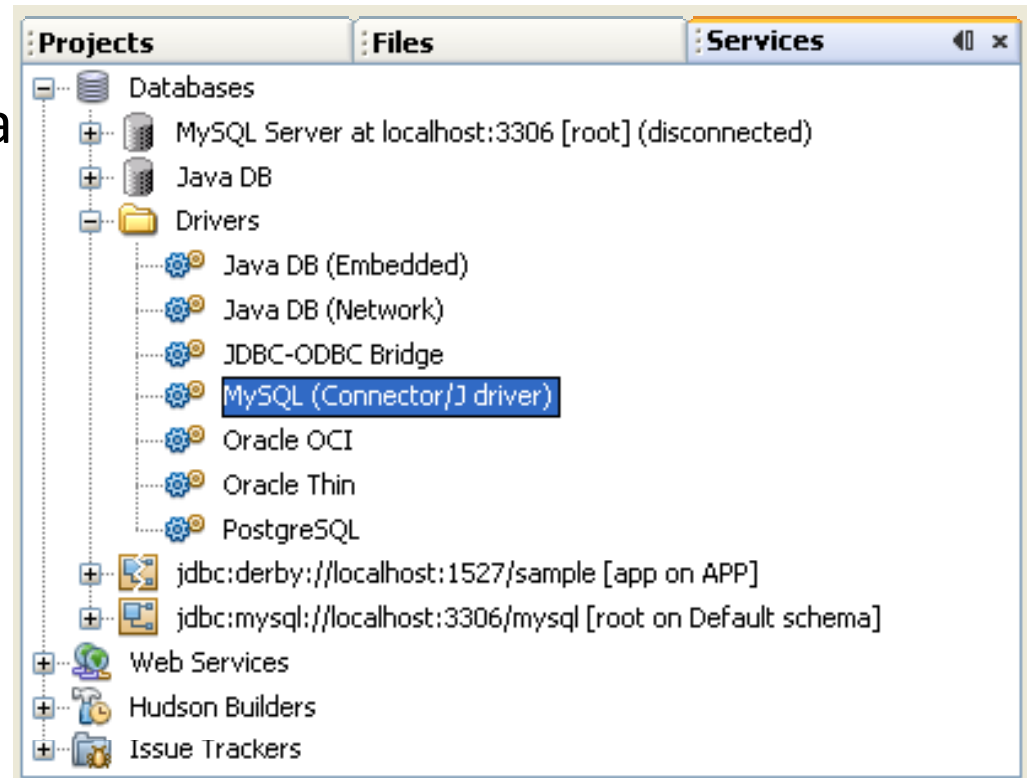
---

- 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:

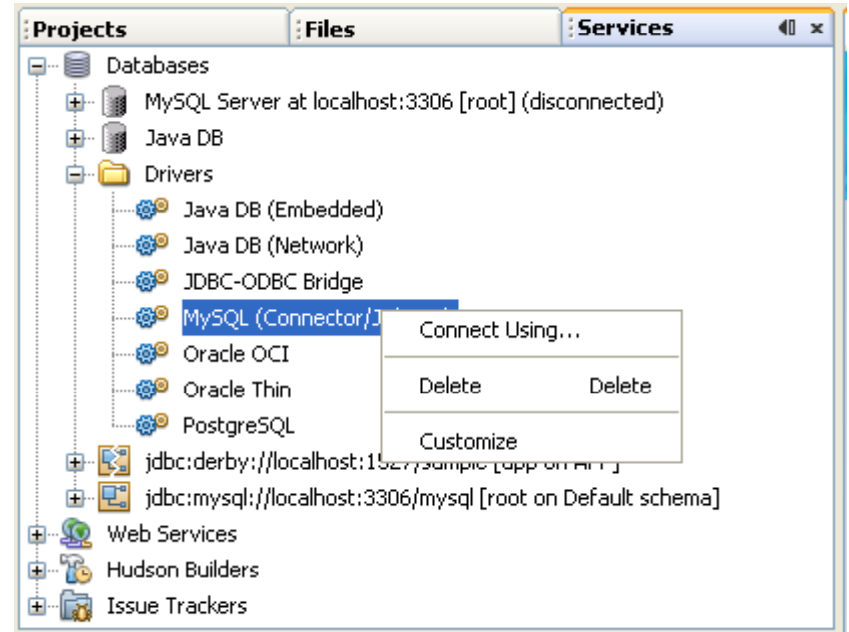


- Expand the Databases item to see a Java DB item, and a Drivers section:
- The idea is that you start the Java DB virtual server, and then create and manipulate databases on the server.



# 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

The screenshot shows a 'New Connection Wizard' dialog box with a blue title bar and a close button. The main area is titled 'Customize Connection' and contains several input fields and a button. The 'Driver Name' is set to 'MySQL (Connector/J driver)'. The 'Host' is 'localhost' and the 'Port' is '3306'. The 'Database' is 'mysql'. The 'User Name' is 'root' and the 'Password' is masked with dots. There is a 'Remember password' checkbox which is unchecked. A 'Test Connection' button is present. The 'JDBC URL' is 'jdbc:mysql://localhost:3306/mysql'. At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**New Connection Wizard**

**Customize Connection**

Driver Name: MySQL (Connector/J driver)

Host: localhost Port: 3306

Database: mysql

User Name: root

Password: ●●●●●●●●

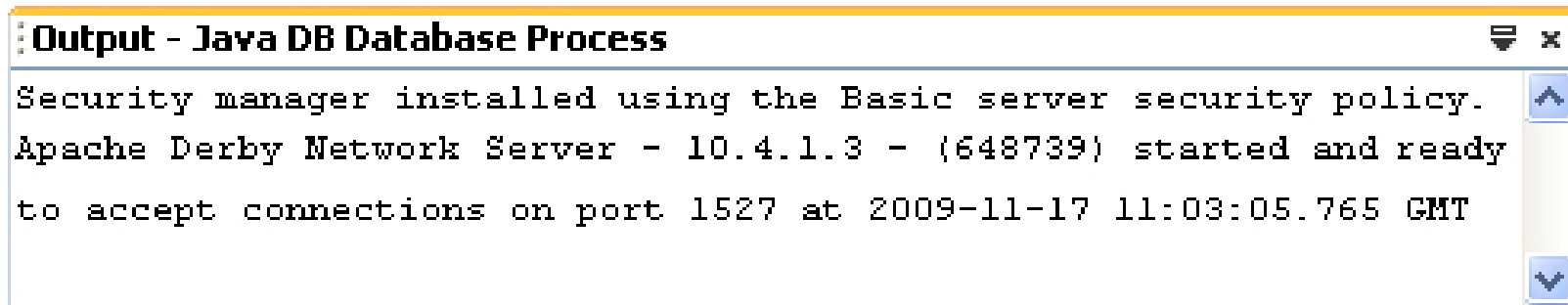
Remember password

Test Connection

JDBC URL: jdbc:mysql://localhost:3306/mysql

< Back Next > Finish Cancel Help

- 
- 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.)



```
Output - Java DB Database Process
Security manager installed using the Basic server security policy.
Apache Derby Network Server - 10.4.1.3 - (648739) started and ready
to accept connections on port 1527 at 2009-11-17 11:03:05.765 GMT
```

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



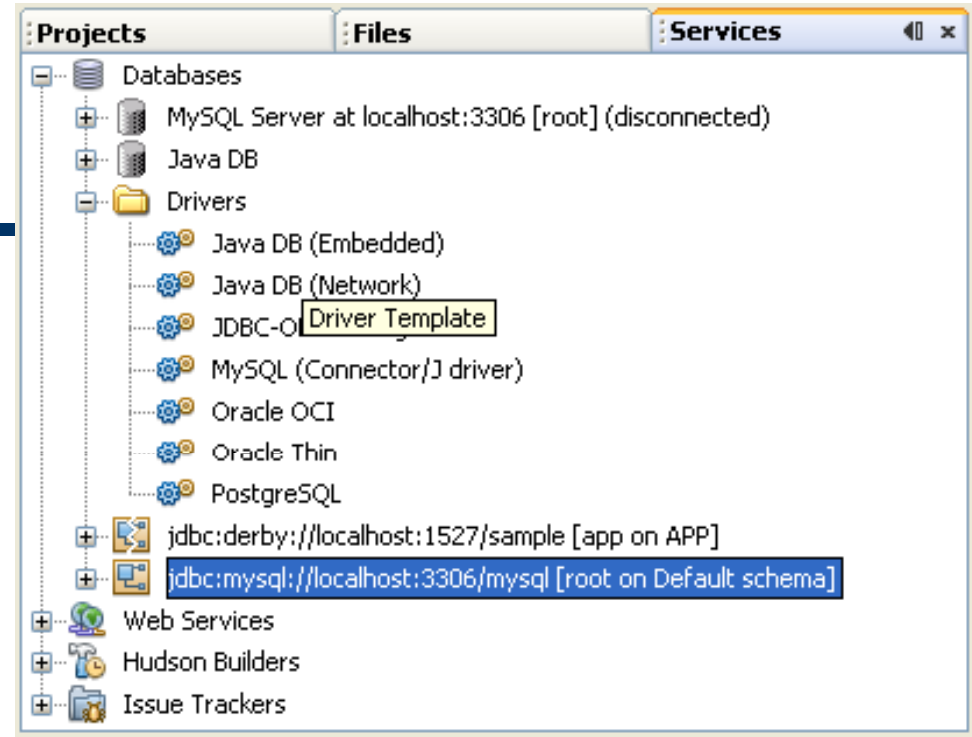
- Press finish

The screenshot shows a 'New Connection Wizard' dialog box with the following fields and controls:

- Driver Name:** MySQL (Connector/J driver) (dropdown menu)
- Host:** localhost (text field)
- Port:** 3306 (text field)
- Database:** mysql (text field)
- User Name:** root (text field)
- Password:** [masked with dots] (password field)
- Remember password (checkbox)
- Test Connection** (button)
- JDBC URL:** jdbc:mysql://localhost:3306/mysql (text field)
- Connection Succeeded.** (information icon and text)

At the bottom of the dialog, there are five buttons: < Back, Next >, Finish, Cancel, and Help.

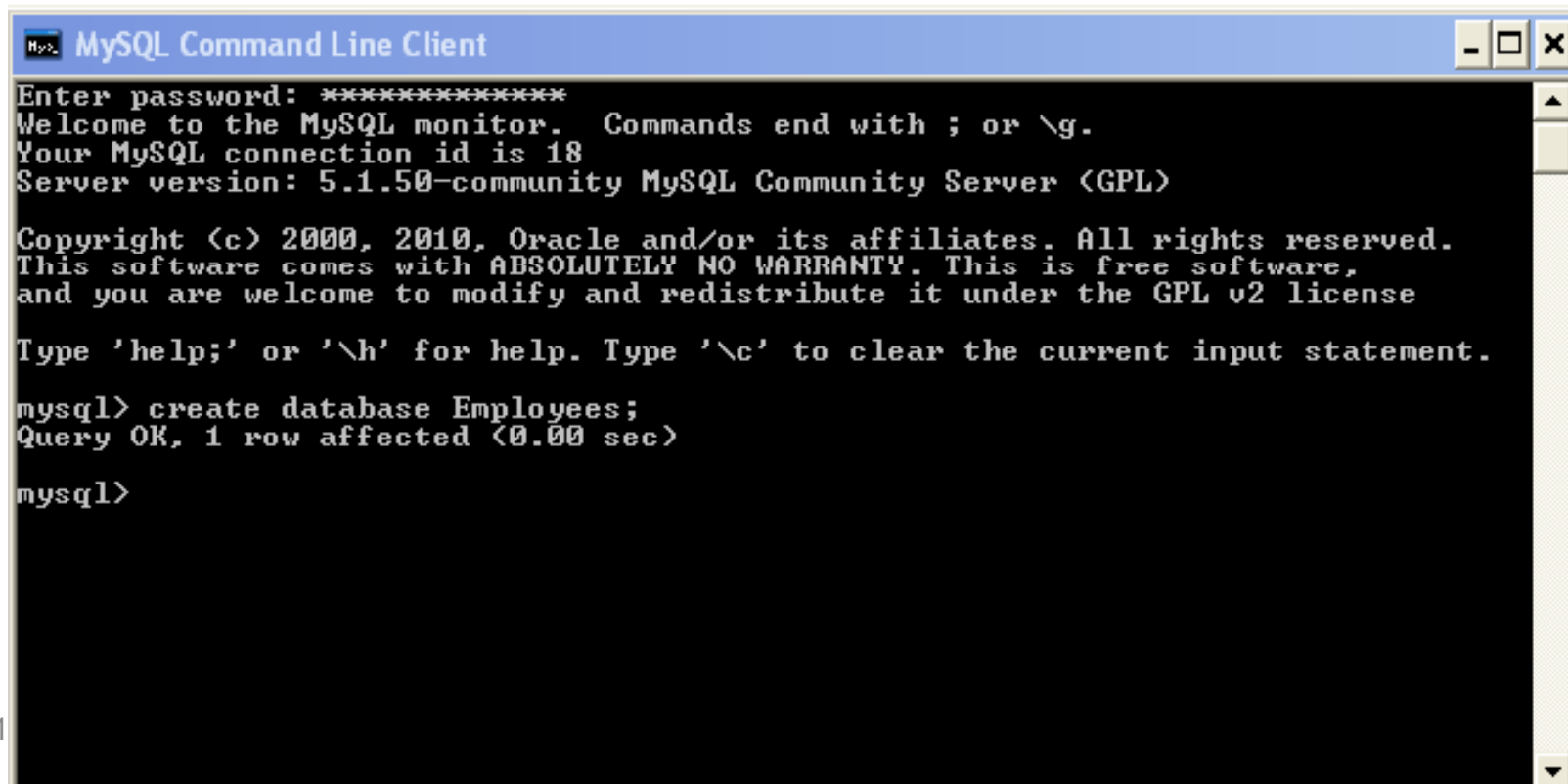
- The connection will appear in the DB tree
- Now you can connect and disconnect to DB by:
- Right click on the connection, select **connect** or **disconnect**
- If you choosed not to remember the password, a dialog box will appear to enter password



# Create new DB with MySql (connector/J driver)

---

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



```
MySQL Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.1.50-community MySQL Community Server (GPL)

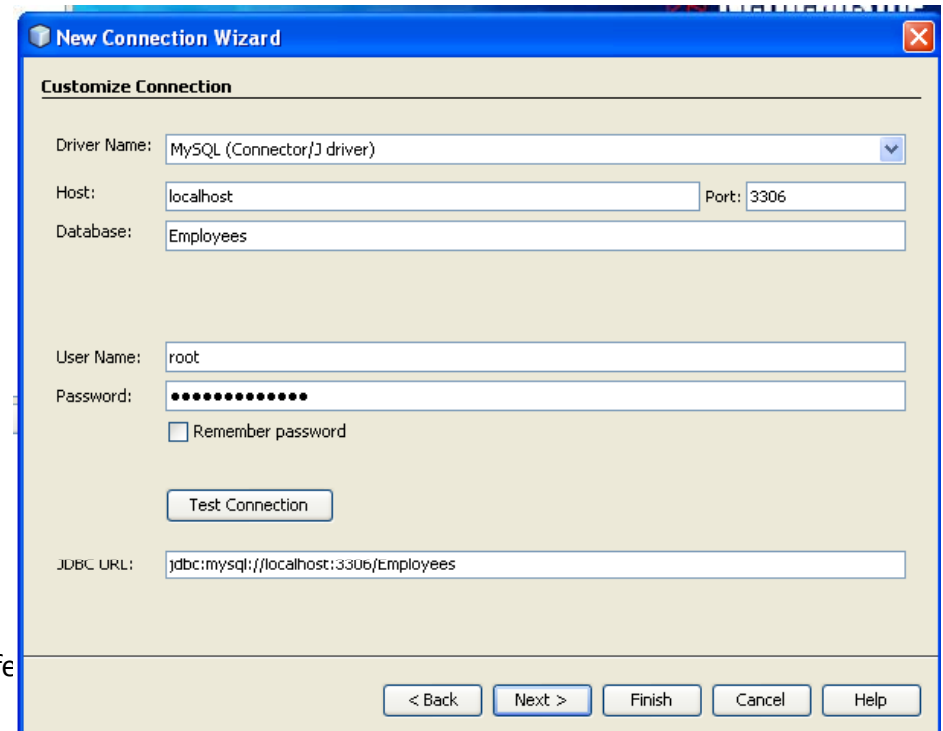
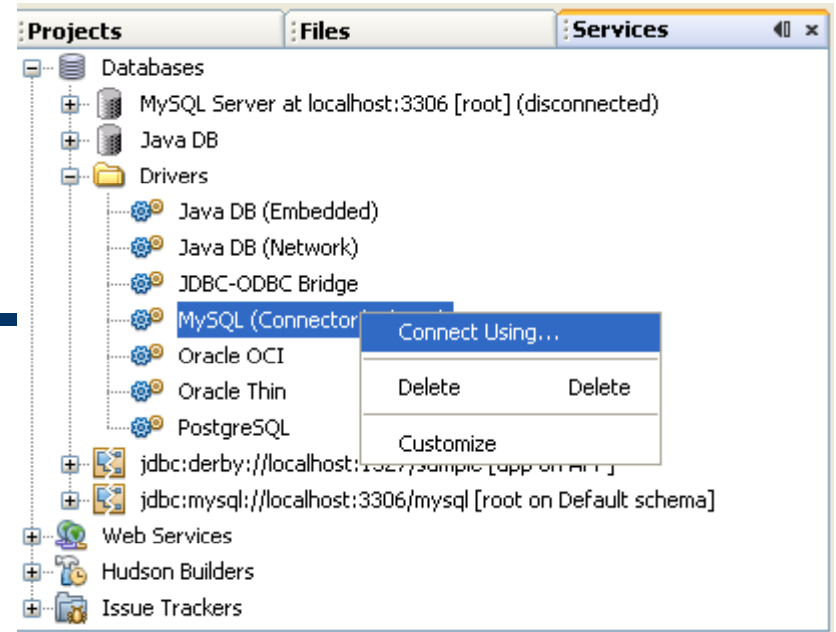
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

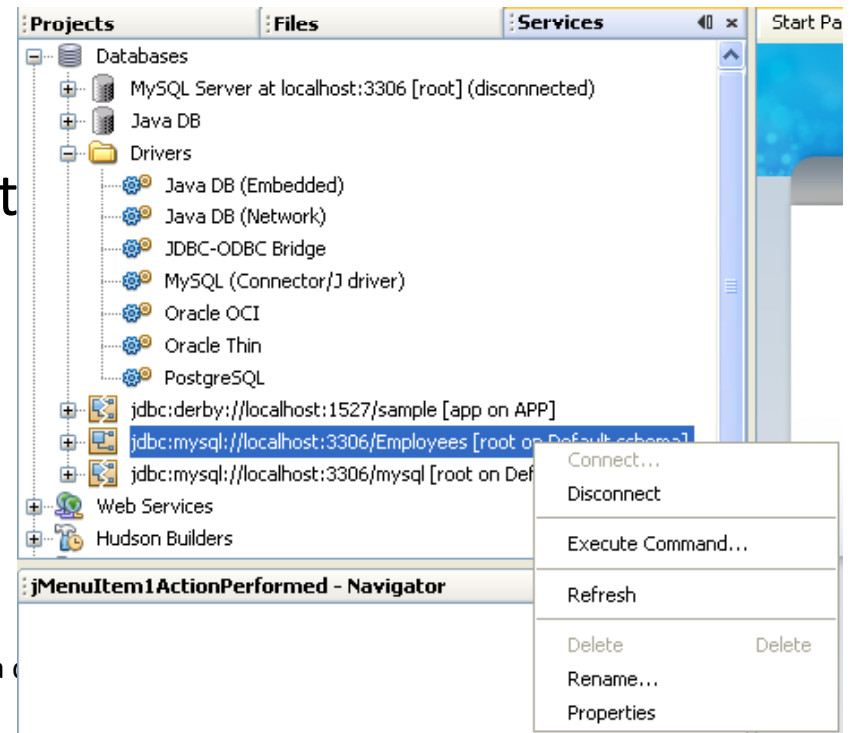
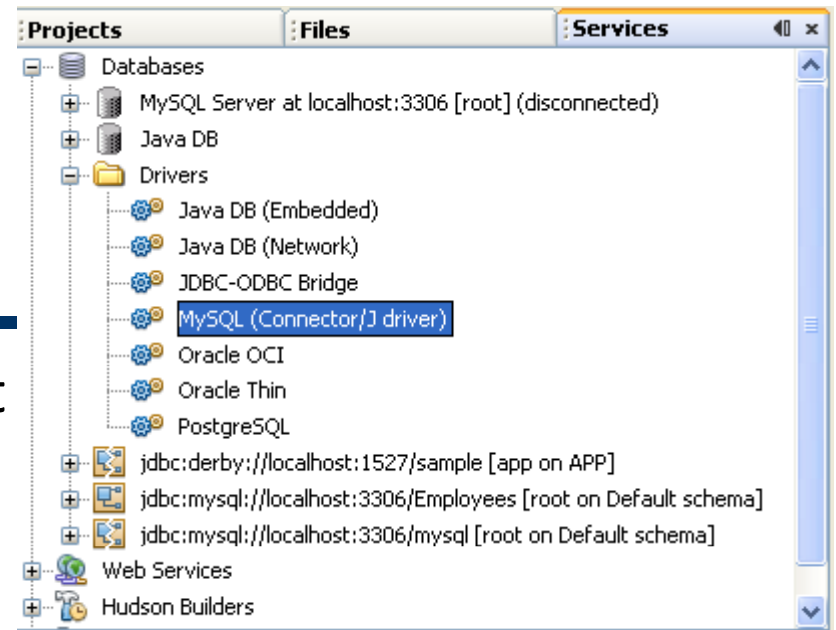
mysql> create database Employees;
Query OK, 1 row affected (0.00 sec)

mysql>
```

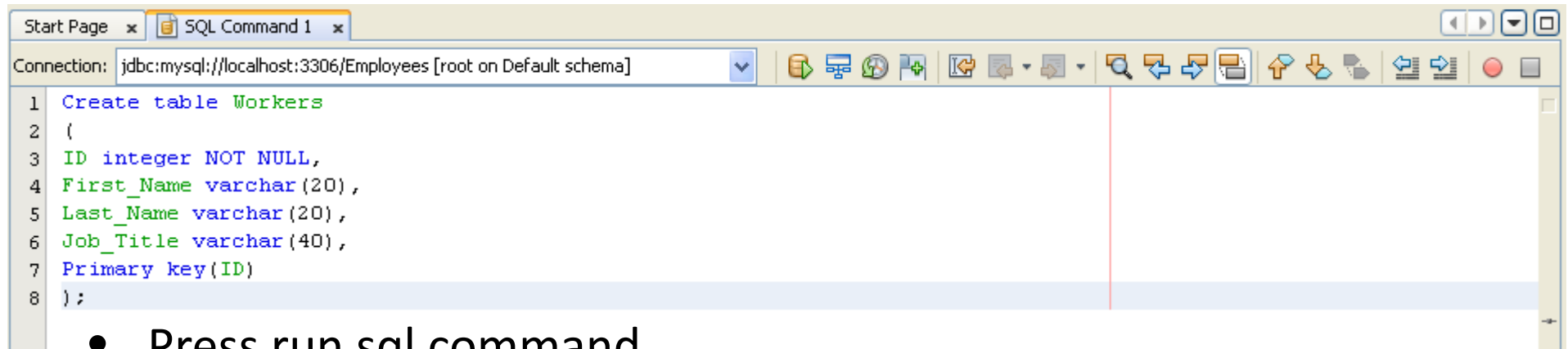
- Return to NetBeans
- Create a new connection to Employees db.
- Change Database name to **Employees**
- Type password, press test connection.
- If every thing is ok,
- Check remember password
- press finish.



- 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:**



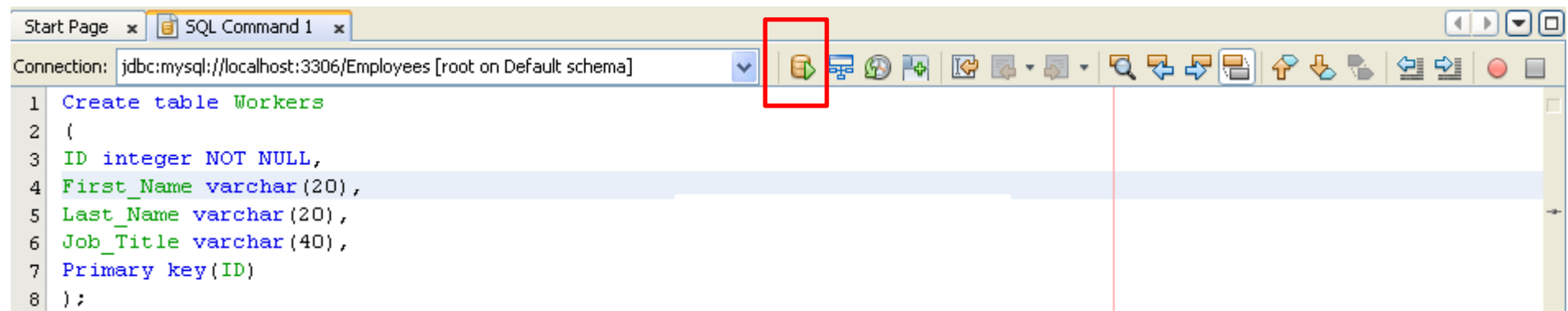
- A command windows will appear.
- Type the following



The screenshot shows a SQL Command window with the following SQL code:

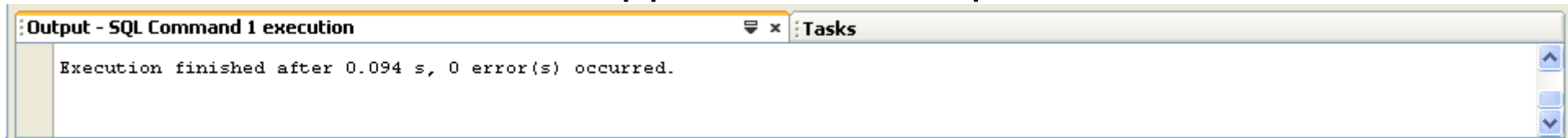
```
1 Create table Workers
2 (
3 ID integer NOT NULL,
4 First_Name varchar (20) ,
5 Last_Name varchar (20) ,
6 Job_Title varchar (40) ,
7 Primary key (ID)
8 );
```

- Press run sql command

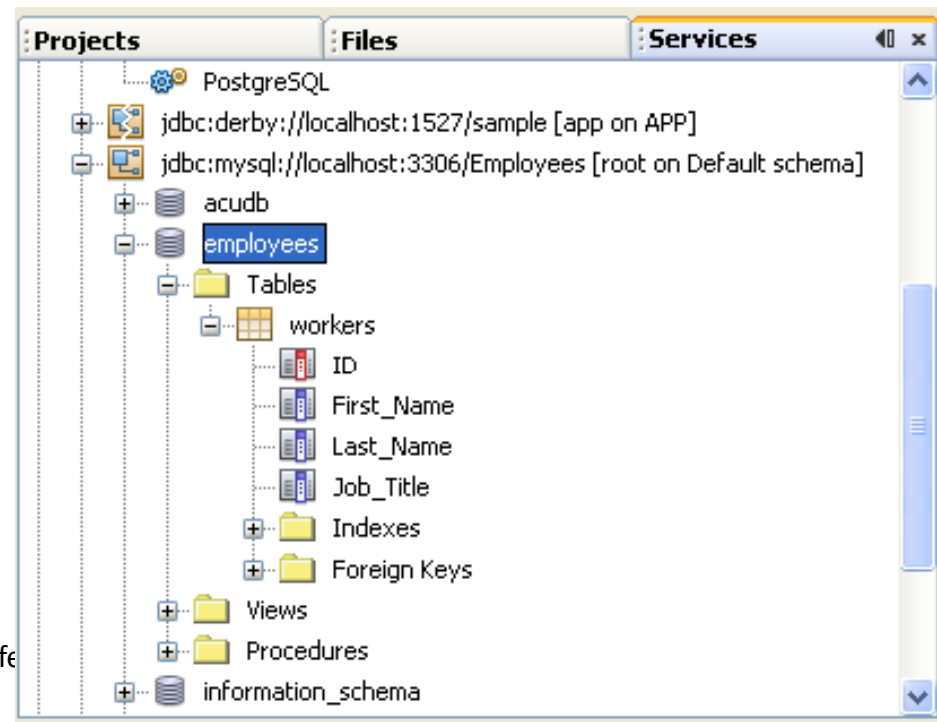


The screenshot shows the same SQL Command window as above, but with the 'Run' button (a green play icon) in the toolbar highlighted by a red box.

- Execution result will appear in the Output window

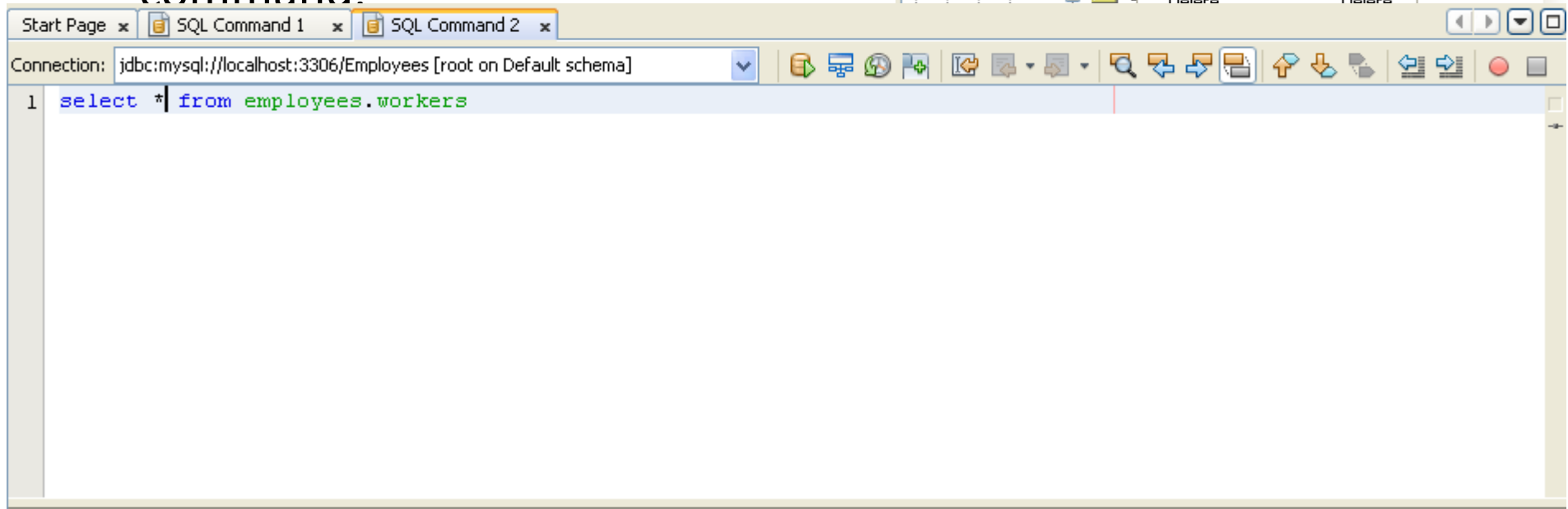
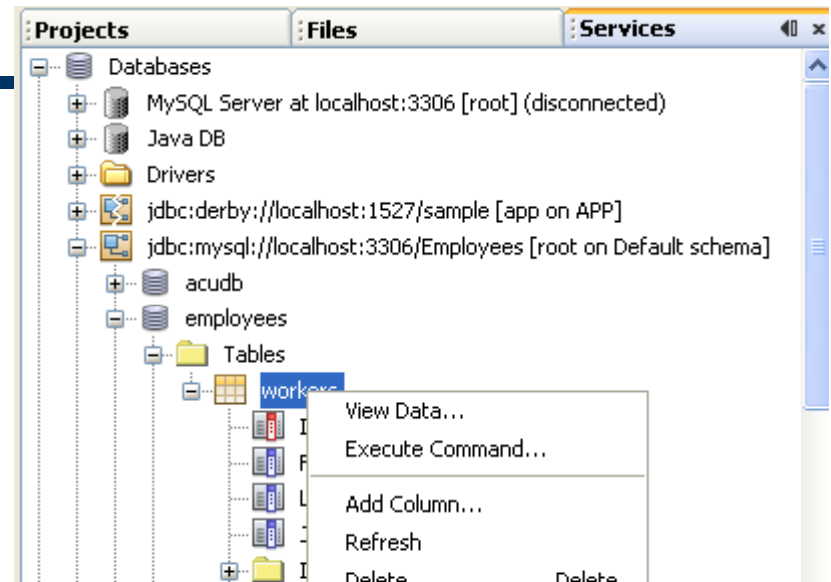


- Table and its columns will appear in the services window



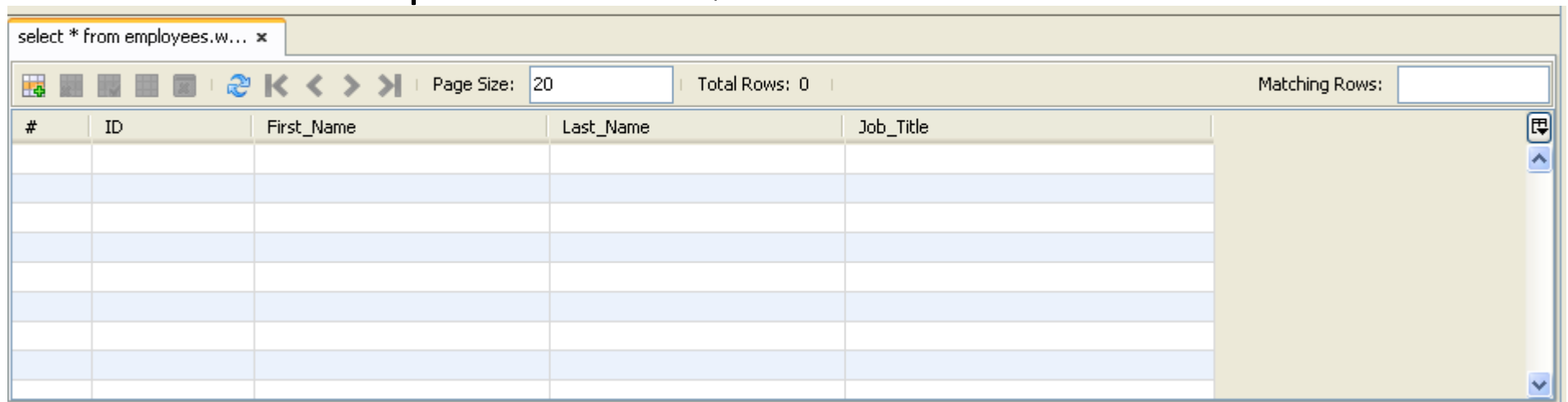
# Adding Records to the Database Table

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





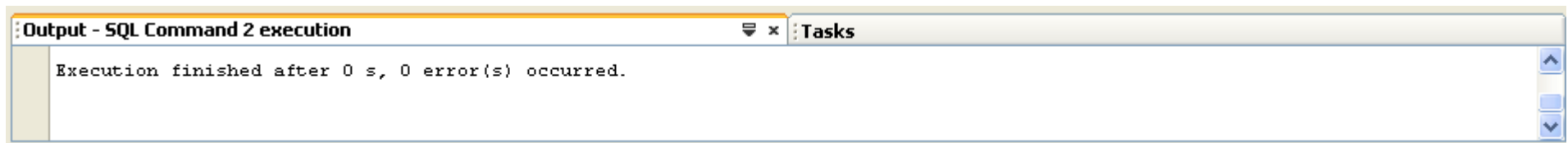
- You use the bottom half of the window to enter new table rows. The top half is for SQL Commands.



The screenshot shows a window titled "select \* from employees.w...". The window has a toolbar with navigation icons and a "Page Size" of 20. The table below has the following columns: #, ID, First\_Name, Last\_Name, and Job\_Title. The table is currently empty.

#	ID	First_Name	Last_Name	Job_Title

The output window contains sql command execution result.



The screenshot shows an output window titled "Output - SQL Command 2 execution". The window contains the text: "Execution finished after 0 s, 0 error(s) occurred."

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



The screenshot shows a database table interface. At the top, there is a toolbar with several icons. The first icon, which is a grid with a green plus sign, is circled in red. To the right of the toolbar, there is a 'Page Size' dropdown menu set to '20'. Below the toolbar is a table with four columns: 'ID', 'FIRST\_NAME', 'LAST\_NAME', and 'JOB\_TITLE'. The table is currently empty, with a light blue shaded area at the bottom indicating a new row being added.

ID	FIRST_NAME	LAST_NAME	JOB_TITLE



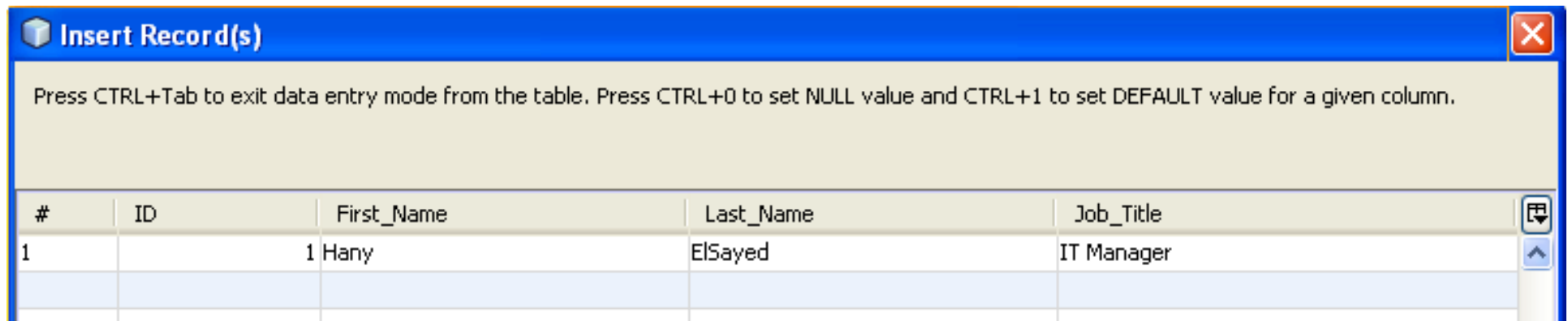
- 
- Enter the following data as the first row of your table:

**ID: 1**

**First Name: Hany**

**Last Name: ElSayed**

**Job Title: IT Manager**



#	ID	First_Name	Last_Name	Job_Title
1	1	Hany	ElSayed	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

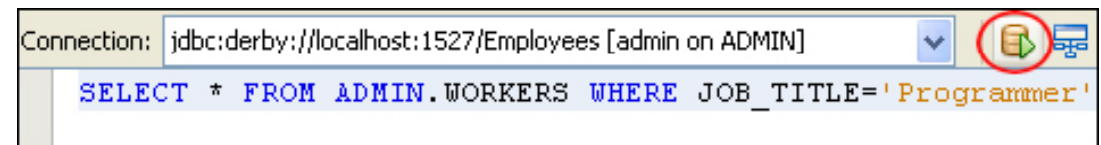
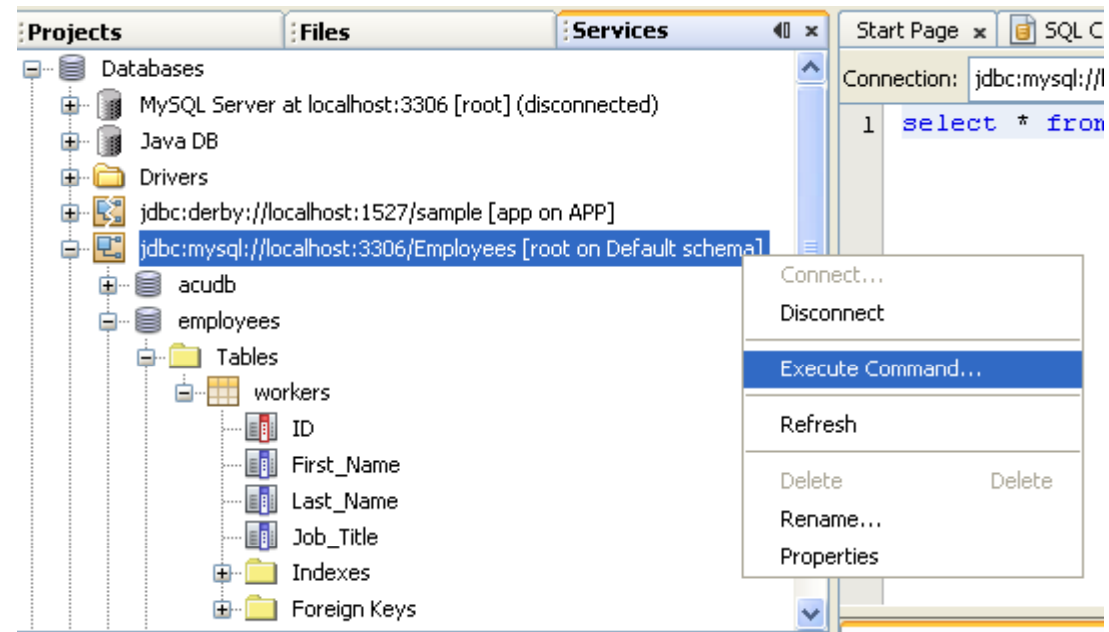
select \* from employees.w... \*

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

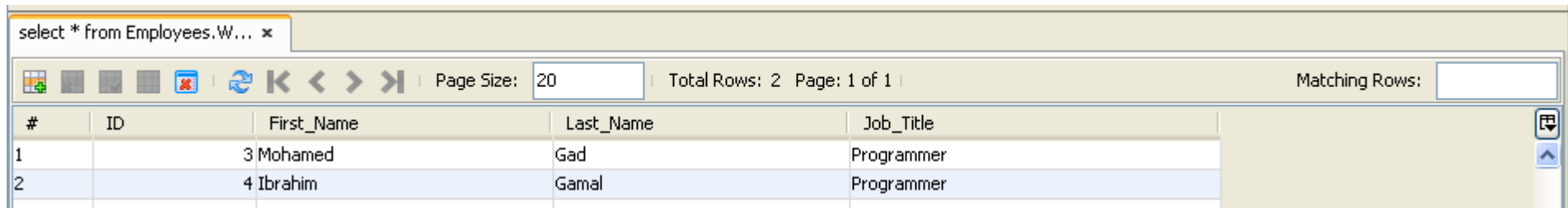
#	ID	First_Name	Last_Name	Job_Title
1	1	Hany	ElSayed	IT Manager
2	2	Mostafa	Hussein	System Analyst
3	3	Mohamed	Gad	Programmer
4	4	Ibrahim	Gamal	Programmer

# 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:



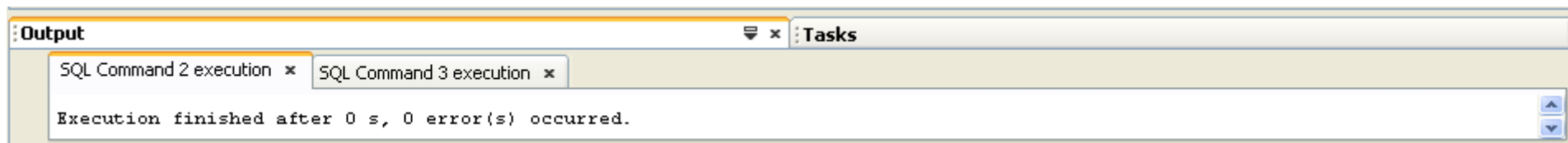
- 
- The results will be displayed in the bottom half of the window:



The screenshot shows a SQL query results window. The query is "select \* from Employees.W...". The results are displayed in a table with the following columns: #, ID, First\_Name, Last\_Name, and Job\_Title. The table contains two rows of data.

#	ID	First_Name	Last_Name	Job_Title
1	3	Mohamed	Gad	Programmer
2	4	Ibrahim	Gamal	Programmer

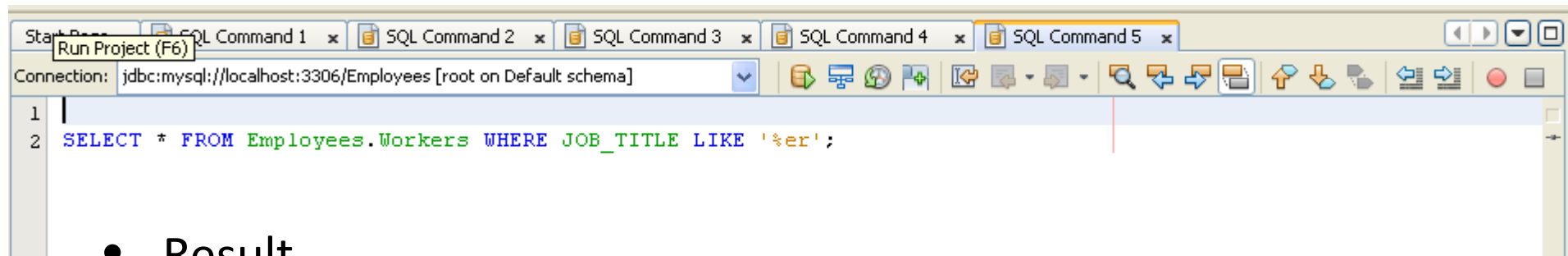
- The output window



The screenshot shows the Output window with the following text:

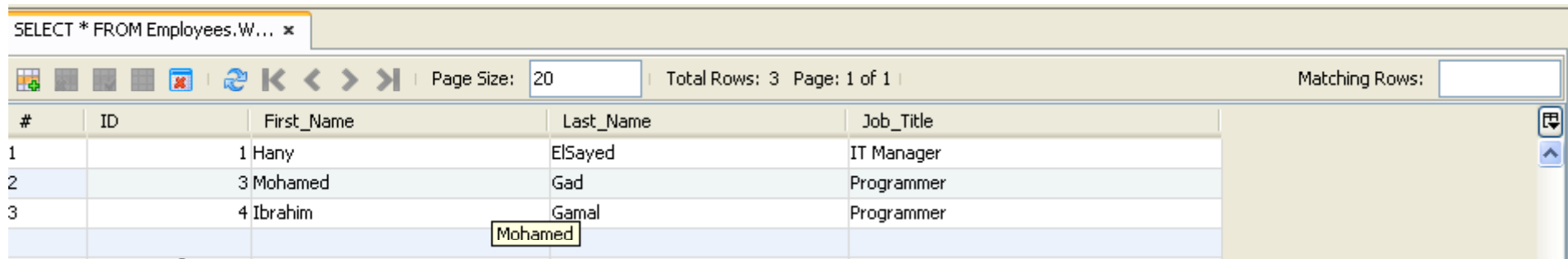
```
SQL Command 2 execution x SQL Command 3 execution x  
Execution finished after 0 s, 0 error(s) occurred.
```

- Try this command too



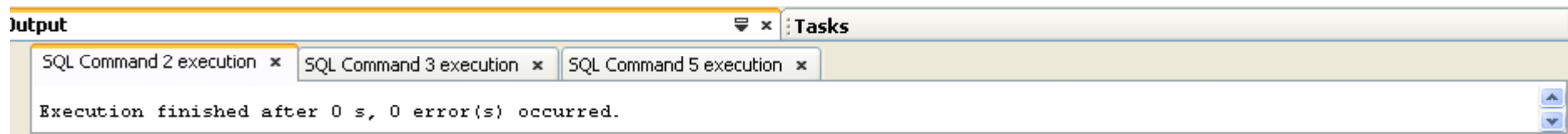
```
1 |
2 | SELECT * FROM Employees.Workers WHERE JOB_TITLE LIKE '%er';
```

- Result



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

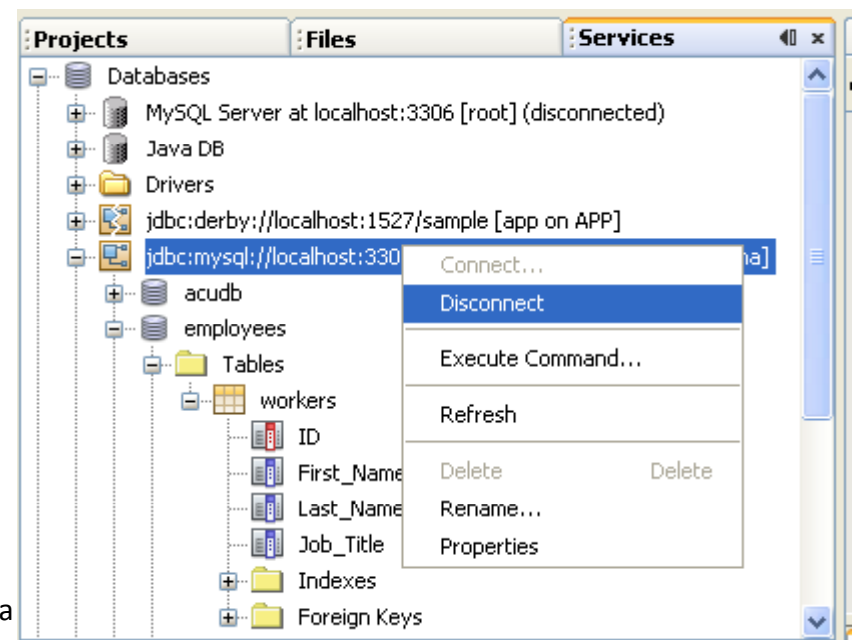
- Output



Output Tasks  
SQL Command 2 execution x SQL Command 3 execution x SQL Command 5 execution x  
Execution finished after 0 s, 0 error(s) occurred.

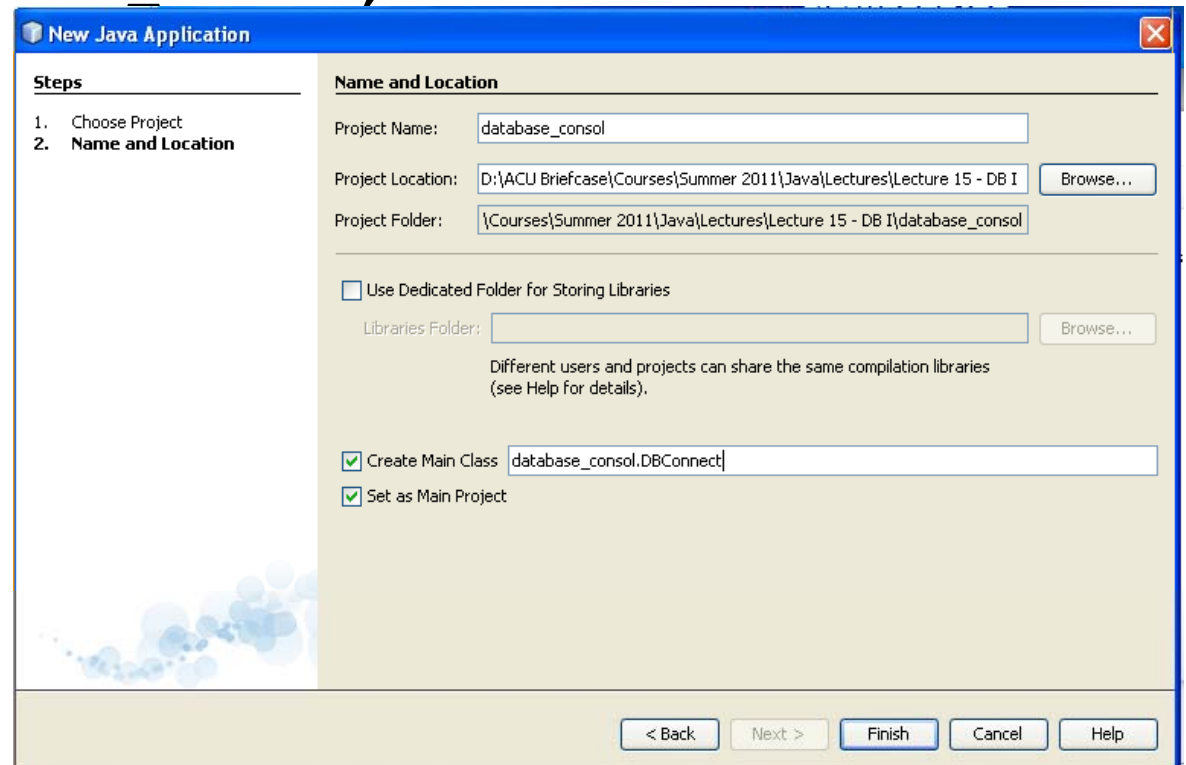


- We'll leave SQL Statements there, as we now have enough to start programming.
- Before doing so, however, close down the connection to your table by right clicking it in the Services are.
- From the menu, select **Disconnect**:



# 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 {

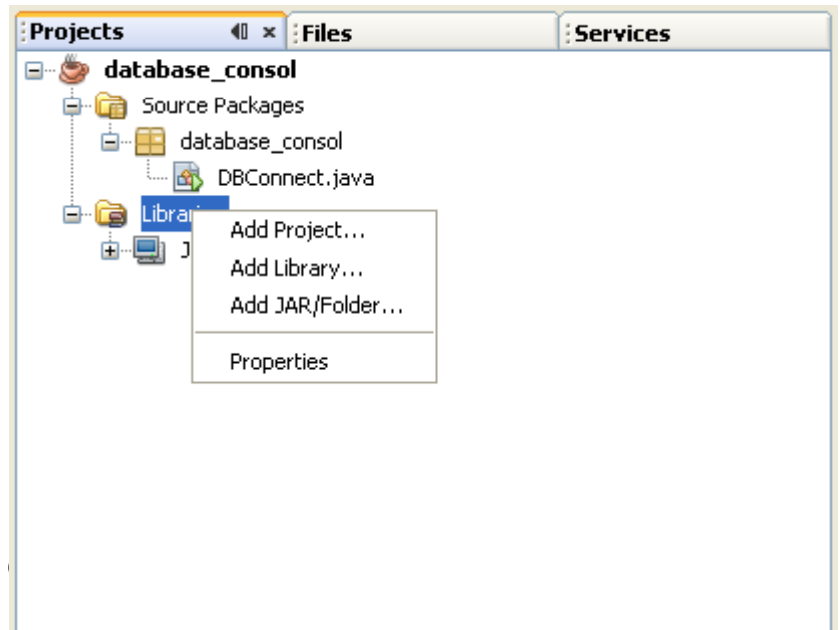
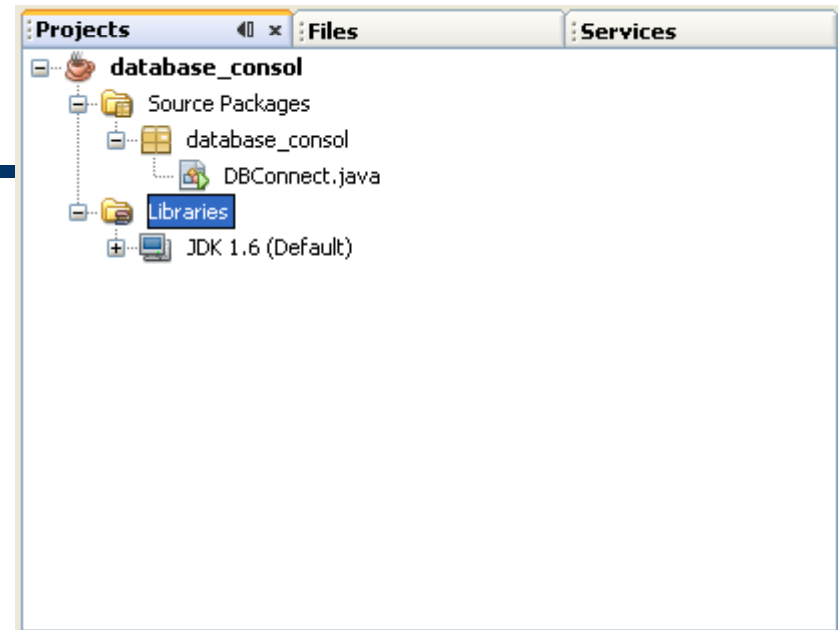
    public static void main(String[] args) {

    }

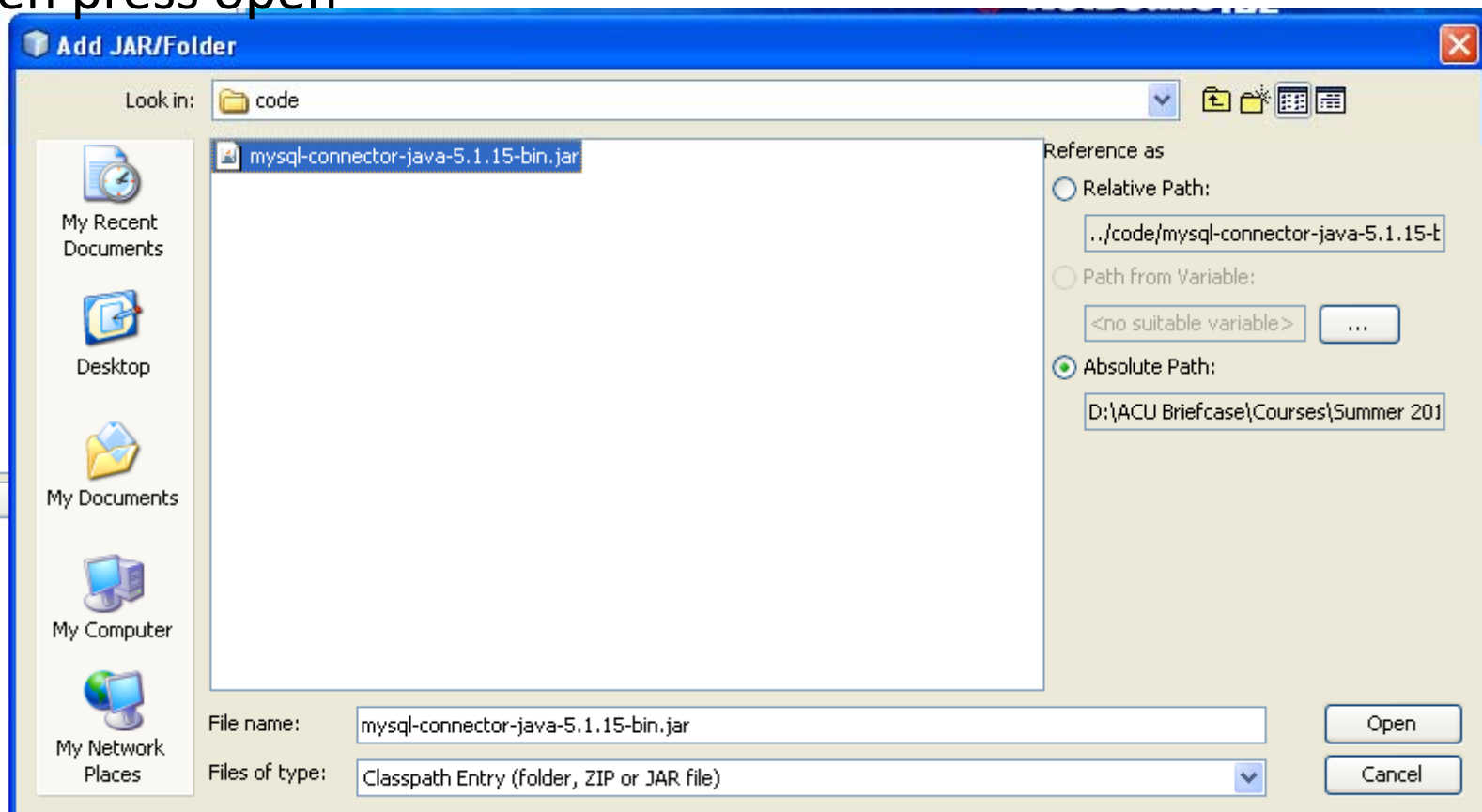
}
```

- 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.

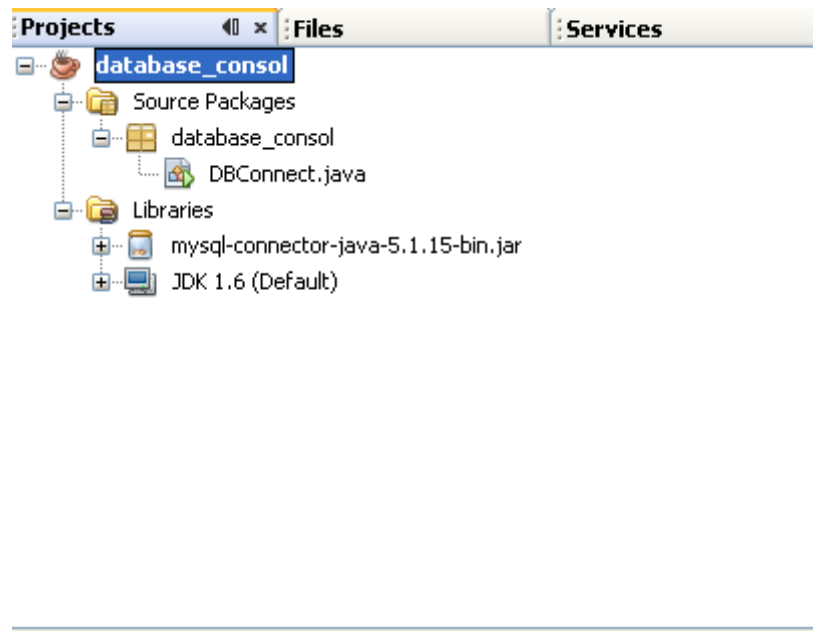
- Now we need to add MySQL j connector library.
- It's a jar file contains necessary API to connect to MySQL database server.
- In **projects** window, open **libraries**
- Right click on libraries and select add jar/folder



- 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:

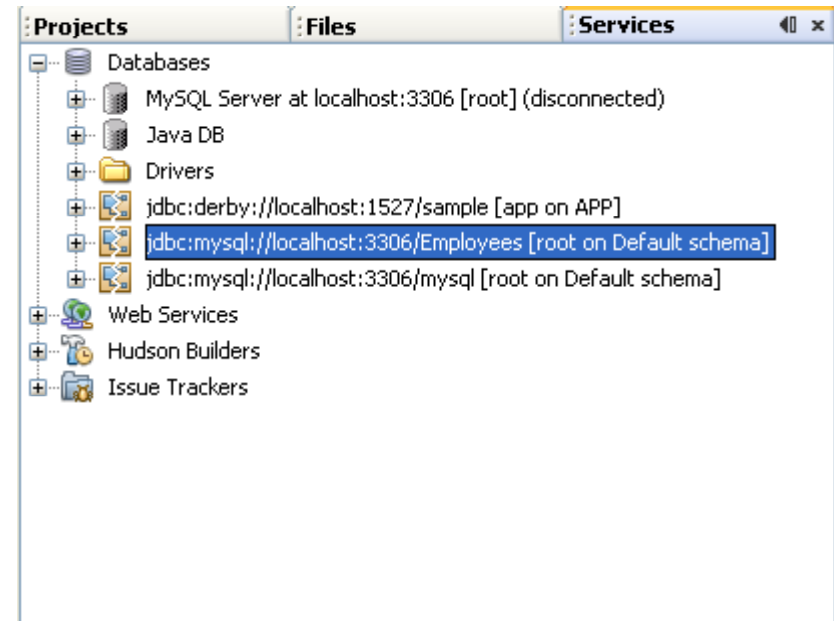
```
Connection con = DriverManager.getConnection( host,  
username, password );
```

- 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

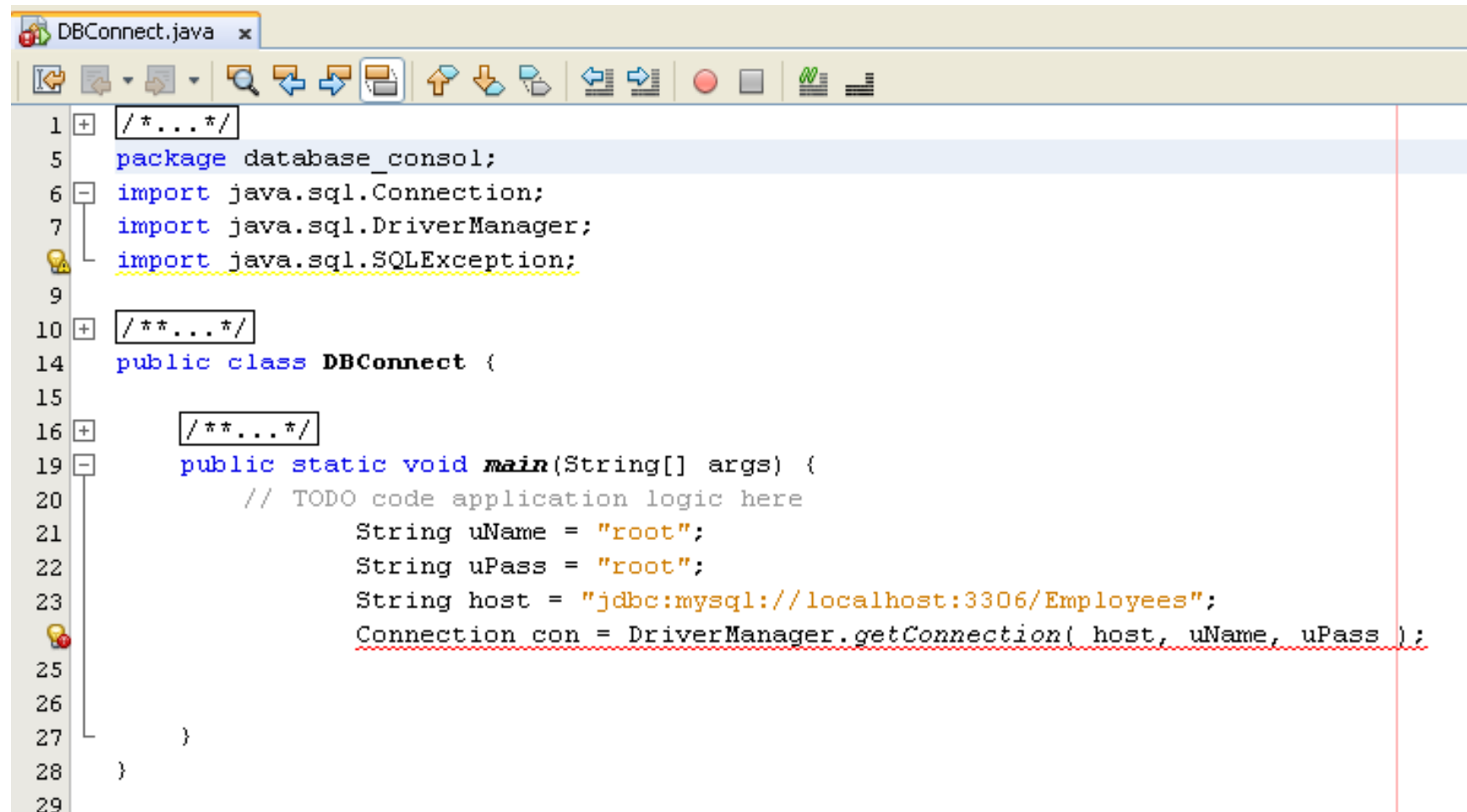
```
String url = "jdbc:mysql://localhost:3306/Employees";
```

- 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 x
1  +  /*...*/
5  package database_consol;
6  -  import java.sql.Connection;
7  |  import java.sql.DriverManager;
8  |  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";
24         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:

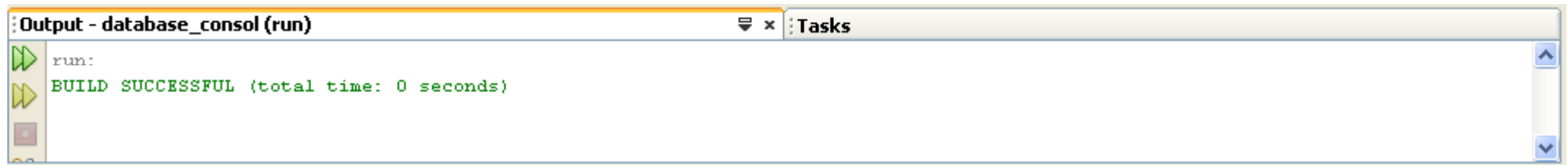
```

+  /**...*/
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());
    }
}
}

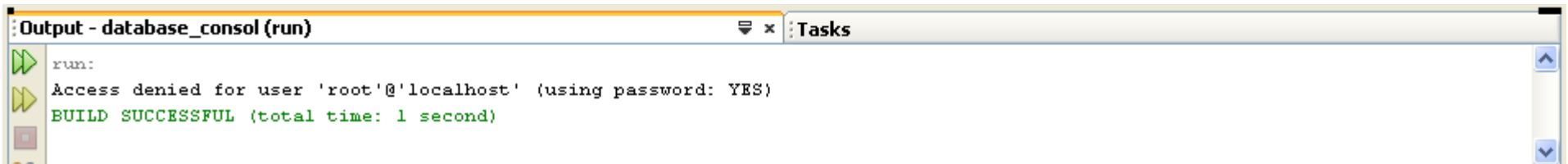
```

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



The screenshot shows an IDE output window titled "Output - database\_consol (run)". The output text is: "run: BUILD SUCCESSFUL (total time: 0 seconds)". The window also has a "Tasks" tab visible on the right.

- If error occurred you will got error message



The screenshot shows an IDE output window titled "Output - database\_consol (run)". The output text is: "run: Access denied for user 'root'@'localhost' (using password: YES) BUILD SUCCESSFUL (total time: 1 second)". The window also has a "Tasks" tab visible on the right.

# 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 **createStatement** 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 **ResultSet** 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( ) ) {  
    }  
}
```



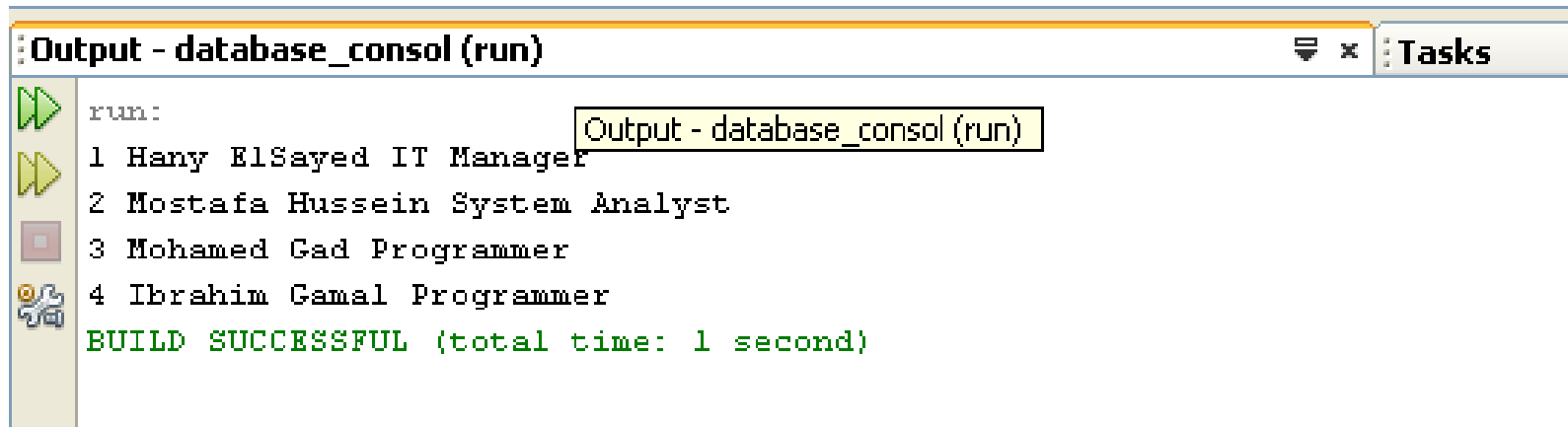
---

## 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)
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

