

# ece-database-2023 LAB 4

---

## Java Database Connectivity (JDBC)

The goal of the lab is to write a utility class named `DataAccess` that enables an application to transparently access its data, using high-level Java methods. The `DataAccess` class is meant to hide from the application how its data is actually stored (here, in a relational database) as well as all the complex machinery required to access it (here, JDBC library and SQL statements).

The role of the application is played by the main method of some other class, e.g. `Test`. When launched, the application first creates an object of the `DataAccess` class. It then uses this object to read or write data to the database using the high-level methods developed in the exercises below.

We take as an example the Company database of lab 2. The `DataAccess` class provides methods for retrieving the employee list, update their salary and so forth.

Please refer to the Java API documentation for more information about the JDBC classes we'll be using: <http://docs.oracle.com/javase/8/docs/api/>. They all belong to the ***java.sql*** package.

## Preamble

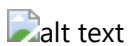
You may use the IDE of your choice to develop the application, e.g. NetBeans, Eclipse, etc. You must first create a plain Java project. Then you need to set up your execution environment by performing the following steps.

## Requirements and Configurations

### Installing the MySQL driver

MySQL's JDBC driver is named "**com.mysql.jdbc.Driver**". It is shipped in a jar file named "**mysql-connector-java-x.y.z.jar**", which you can download from Campus. You must add this jar file to the libraries of your project: this will include the jar file in the classpath of the project.

you can download the latest JDC connector here : <https://dev.mysql.com/downloads/connector/j/>



Right click on "Libraries" -> Add JAR/Folder -> Select your JDC connector x.x.xx

### Next Step :

1. Go to service




Right click on "Databases"

|

v

New connection

2. add driver

 alt text

Select "MySQL (Connector/J driver)"

|

v

Add

|

v


Select your JDC connector x.x.xx

|

v

click on Next

3. configure connection

 alt text

Change the Database name in our case is "company"

|

v

click on "Test Connection"

|

v

if "Connection Succeeded" Click on next

|

v

next

|

v

finish

## Setting the name of the driver

Next, you have to tell the JDBC runtime which driver you want to use. This is done through the `jdbc.drivers` system property, which you must set to the name of the driver.

You can do so by executing the following instruction in your code:

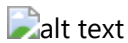
```
System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");
```

Better yet, you can set the `jdbc.drivers` property as a VM option when launching the application. If you launch it from the command line, you must use the following syntax:

```
java -Djdbc.drivers=com.mysql.jdbc.Driver <class name>
```

If you launch the application from your IDE, you must set the system property in the “VM options” section of the “Run” configuration of your project, as follows:

```
-Djdbc.drivers=com.mysql.jdbc.Driver
```



## Setting the database's URL

The JDBC URL format for the MySQL's driver is as follows :

```
jdbc:mysql://[host][:port]/[database]
```

Use the values defined in the “How to connect to your database” tutorial for the host, port and database parts of the URL.

## Miscellaneous

Development tip: to avoid writing ***try/catch*** clauses everywhere in your code, add the ***throws SQLException*** clause to all the methods / constructors that you develop, including main.

## Exercise 1

Create Connection with MySQL :

The constructor takes the database's **URL**, the **user's login** and **password** as parameters and establishes a connection to the database. The connection is stored in a (private) instance field, since all the other methods of the class use the connection

### DataAccess.java

```
package model;

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

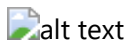
```
import java.sql.DriverManager;

public class DataAccess {

    private Connection connection;

    public DataAccess(String url, String login, String password) throws
        SQLException {
        connection = DriverManager.getConnection(url, login, password);
        System.out.println("connected to " + url);
    }
}
```

When using NetBeans, you can set these arguments in the **"Arguments"** section of the **"Run"** configuration of your project: in the corresponding field, list all the arguments on a single line, separated with space characters.



### Workaround for NetBeans Bug

The parameters of the constructor are taken from the ☐ **args** parameter of the main method.

```
public class Test {
    /**
     * @param args the command line arguments
     */
    * @throws java.lang.Exception
    */
    public static void main(String[] args) throws Exception {
        ...
    }
}
```

There is a conditional check at the beginning of the main method :

```
if (args.length == 2) {
    args = Arrays.copyOf(args, 3);
    args[2] = "";
}
```

This code checks if there are exactly two command-line arguments (`args.length == 2`). If there are only two arguments, it adds an empty string as the third argument. This is done as a workaround for a potential bug in NetBeans, which might not pass the third argument correctly in certain situations.

### Creating a Data Access Object

After the workaround, the code proceeds to create an instance of the `DataAccess` class:

```
// create a data access object
data = new DataAccess(args[0], args[1], args[2]);
```

It uses the command-line arguments (args) as parameters to the DataAccess constructor. The assumption here is that the first argument is the database URL, the second argument is the username, and the third argument is the password for the database connection.

### Test.java

```
package test;

import java.util.Arrays;
import model.DataAccess;
/**
 *
 * @author Jean-Michel Busca
 */
public class Test {
    /**
     * @param args the command line arguments
     *
     * @throws java.lang.Exception
     */
    public static void main(String[] args) throws Exception {
        // work around Netbeans bug
        if (args.length == 2) {
            args = Arrays.copyOf(args, 3);
            args[2] = "";
        }

        // create a data access object
        data = new DataAccess(args[0], args[1], args[2]);

        // access the database using high-level Java methods
        // ...
        // close the data access object when done
    }
}
```

### Output :

```
run:
connected to jdbc:mysql://localhost:3306/company
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Close method :

To close a database connection in Java, you should use the **close()** method provided by the **java.sql.Connection interface**, which is typically implemented by database connection classes like **java.sql.Connection**, **java.sql.Statement**, and **java.sql.ResultSet**.

Here's how you can close a database connection in your Java code:

### DataAccess.java

```
// Method to close the database connection
public void closeConnection() {
    if (connection != null) {
        try {
            connection.close();
            System.out.println("Database connection closed.");
        } catch (SQLException e) {
            // Handle any potential exceptions here
            e.printStackTrace();
        }
    }
}
```

After that we need to call the close method from "**DataAccess**" class, as below :

### Test.java

```
public static void main(String[] args) throws Exception {
    DataAccess data = null;

    // work around Netbeans bug
    if (args.length == 2) {
        args = Arrays.copyOf(args, 3);
        args[2] = "";
    }

    try {
        // create a data access object
        data = new DataAccess(args[0], args[1], args[2]);

        // access the database using high-level Java methods
        // ...
    } finally {
        // close the data access object when done
        if (data != null) {
            data.closeConnection();
        }
    }
}
```

```
}
```

**Output :**

```
run:
connected to jdbc:mysql://localhost:3306/company
Database connection closed.
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Exercise 2

Write the method List `getEmployees()` that returns the number, name and salary of all the employee in the EMP table. Note: the class `EmployeeInfo` is already defined in the model package.