JDBC Practical

Objectives

The objective of this practical session is to connect to a database using JDBC, inserting and removing rows, and querying the database.

Reference Material

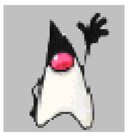
This practical session is based on the material covered in the *Introduction to JDBC* chapter. Additional information can be found in the JDK documentation.

Overview

In this practical you will complete the code of an existing class called `EmployeeApplication` which connects to a MySql database. The application allows employees to be added and removed, in a similar way to an earlier practical. You can test your class with the aid of a graphical user interface, but there is no need to alter the GUI code.

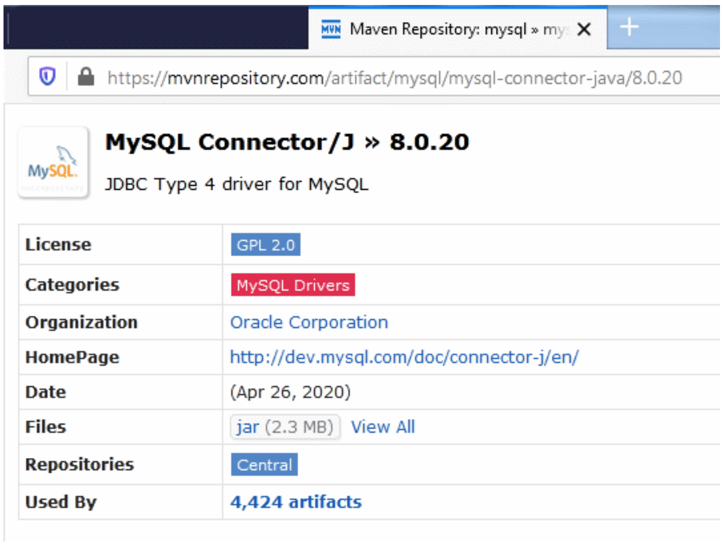
Practical: Connecting to a MySql Database

Setting up the database server and tools



1. The application we will be working on uses a MySql database. Your instructor will have configured an online instance of this database, and will give you the URL. Later, if you would like to do this yourself, there are instructions at the end of this document.
2. Download the JDBC driver for MySql from the URL:

`https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.20`



Maven Repository: mysql » my

https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.20

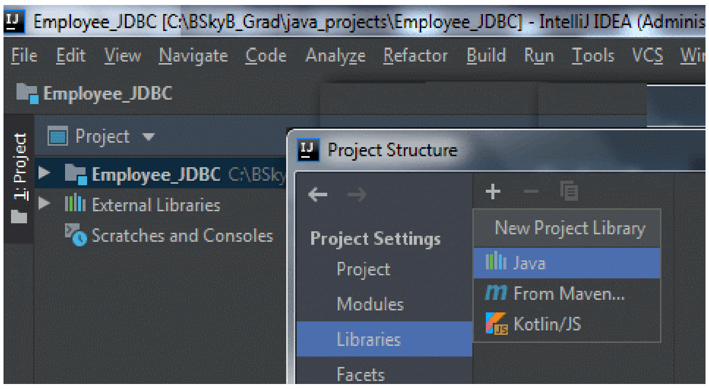
MySQL Connector/J » 8.0.20

JDBC Type 4 driver for MySQL

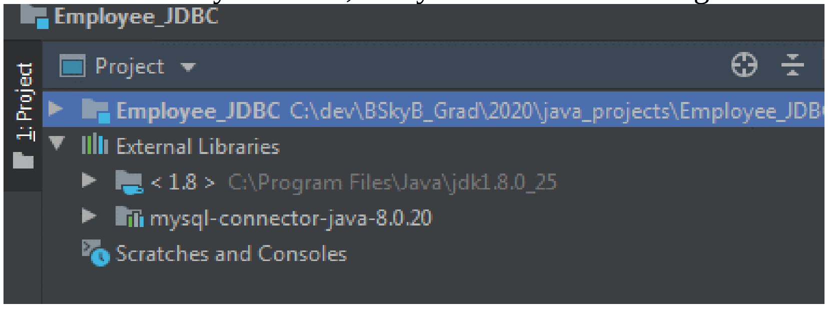
License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Apr 26, 2020)
Files	jar (2.3 MB) View All
Repositories	Central
Used By	4,424 artifacts

Save the JAR file to your computer.

3. You will need this in your classpath for all projects. In IntelliJ, right-click on your project, and **Open Module Settings > Libraries**, and click on the + button on the right-hand side, selecting **Java**:



Browse to the JAR file you just saved. IntelliJ should determine that your driver JAR file really is a JAR, and you can **OK** the dialogs


Employee_JDBC
Project
Employee_JDBC C:\dev\BSkyB_Grad\2020\java_projects\Employee_JDB
External Libraries
< 1.8 > C:\Program Files\Java\jdk1.8.0_25
mysql-connector-java-8.0.20
Scratches and Consoles

4. The JAR file contains classes which implement the JDBC API interfaces, such as `Connection`, `Statement`, etc. You can see these by expanding the file in IntelliJ. Now you can write generic JDBC code to interact with the database.

Practical

Part 1. A JDBC Employee Application

1. Import the source code under `exercises/student/JDBC/` into your project.
2. Begin by examining thoroughly the starter code you have been given. Note the two classes: `EmployeeMain` and `EmployeeFrame`. The `Frame` class contains all the graphical code, including event handlers which call methods in an `EmployeeController` object. Both are created and linked in the `main()` method of `EmployeeMain`. There is an implementation of the interface `EmployeeController`, `EmployeeControllerDummy`, which gives trivial responses and doesn't connect to any database.
3. You plan will be to write a new class `EmployeeControllerJDBC`, which implements the `EmployeeController` interface, to house the database code you need.

4. The only existing code you need to change will be in `EmployeeMain`, to create an instance of your own class instead.
5. Create the new class in your project.
6. To connect to database using JDBC it is necessary to load a JDBC driver into memory and register it with the `DriverManager`. This step only needs to be completed once per-JVM, so we can accomplish the task using a `static` block.
7. A static block is a block of code which is executed only once – on the first occasion when a class is loaded into memory. A static block is placed within the main body of a class definition (just like a static method) and has the following form:

```
static {
     // code goes here
} // end of static block
```

8. Define a static block for the `EmployeeControllerJDBC` class. Within the static block, use the method `Class.forName()` to load the driver “`com.mysql.jdbc.Driver`”. Remember to catch the `ClassNotFoundException`.
9. In the constructor of `EmployeeApplication`, open a database connection, using the method in the `DriverManager` class. The URL you need will be:

   `jdbc:mysql://XX.XX.XX.XX/employeeDB`

10. The username and password are the “`root`” and “” respectively (under the earlier assumption). Store this in the instance variable provided.
11. The database table '`employees`' has the following columns:
    1. `id`, an integer value (`int`)
    2. `firstname`, a `VARCHAR` (`String`)
    3. `lastname`, a `VARCHAR` (`String`)
    4. `age`, an integer value (`int`)
12. Edit the method `getEmployees()` so that it queries the database and builds a `StringBuilder` containing the details of the members, in turn. You will need to create JDBC `Statement` and `ResultSet` objects, using a suitable SQL query. Do not forget to use the `close()` methods, and catch exceptions appropriately! Compile and test you application.
13. In order to be able to add and remove employees, you now turn your attention to the methods `addEmployee()` and `deleteEmployee()`. Just as before, you will need to create a JDBC `Statement` object, but this time use SQL `INSERT` and `DELETE` updates.
14. In your `addEmployee()` method, you will need to concoct an ID number for the employee. There are 3 techniques you could choose:
    1. Create a random number with `Math.random()`, and hope it doesn't clash with another ID in the database.

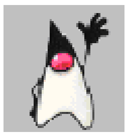
    2. Use `System.currentTimeMillis()` to get a number which you can use directly, or perhaps with some modular operation. Once again, there is a danger this could clash.
    3. Check the database (with a `SELECT` query) to find an unused number. Is this safe?

15. After implementing your code, test your application.
16. Be sure you understand how the `EmployeeFrame` and `EmployeeController` work together so that the JDBC `Connection` opened in the `EmployeeControllerJDBC` constructor always gets closed.
17. Note that this application is written without the use of an `Employee` class. Is this a good idea? What are the pluses and minuses of this approach.

As an extra challenge, you might try to implement `Employee` and `Manager` relationships, as you did in the Inheritance exercise. You will need to alter your database schema, and write SQL to cope with relational joins.

Practical: MySql Set-Up on your machine

Setting up the database server and tools



1. If you have MySql installed your machine, you can connect to that. If you need to start the server, open a terminal and use the command: `mysqld`.
2. Hereafter, we will assume the username is "`root`" and password is "", but this may vary depending on your set-up (password may be "root" in some installations).
3. Open a second terminal, and type in the commands:

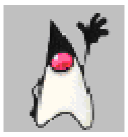
```
mysql –u root

create database employeedb;

use employeedb;

create table employees (id int not null,
      firstname varchar(40), lastname varchar(40),
      age int not null);
```
(all this on one line)

```
insert into employees values (1,'Fred','Bloggs',33);

select * from employees;

\quit
```

4. This should prove that your database works. Now you can try this out with your application. But you must change the URL to point to `localhost` instead of the URL given by your instructor.

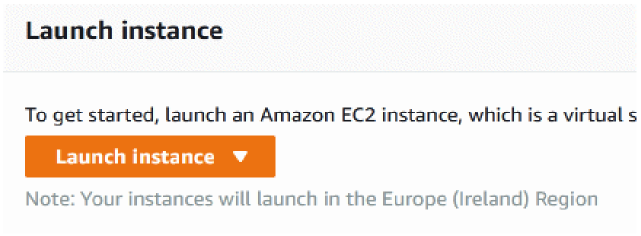
Practical: MySql Set-Up on AWS

Setting up the database server



These instructions are for setting up the database on an AWS Cloud machine. It uses a MySql database pre-packaged in a Docker container. This is most likely what your instructor did to launch the online MySql that you connected to in earlier parts of this exercise.

1. Log-in to your AWS account and locate the **Launch Instance** button for EC2.

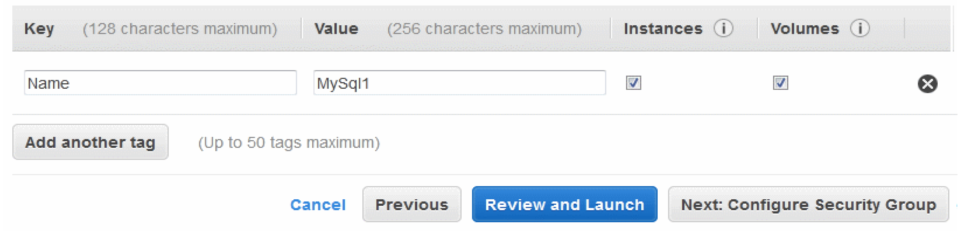


Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual s

Note: Your instances will launch in the Europe (Ireland) Region

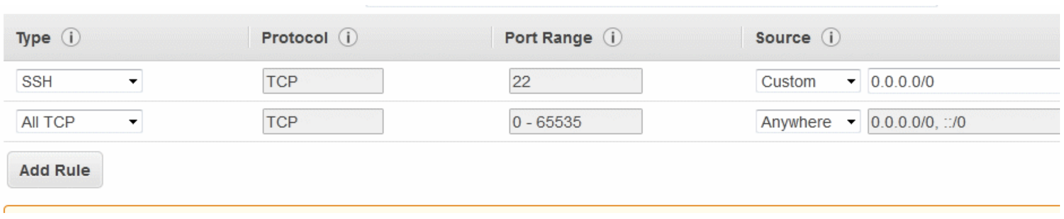
2. Choose **Amazon Linux 2 AMI**, click Select. On the next screen, accept **t2.micro**, and click on **Next: Configure Instance Details**.
3. Accept defaults and then **Add Storage**. Accept defaults and **Next: Add Tags**.
4. Add a tag with key 'Name' and value 'MySql1'. This just helps you locate your machine on the list of running instances.



Key (128 characters maximum)	Value (256 characters maximum)	Instances	Volumes
Name	MySql1	✓	✓

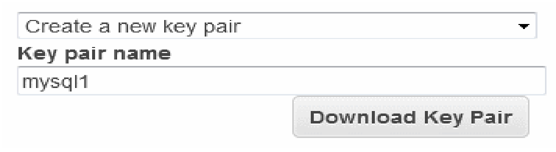
Click **Next: Configure Security Group**.

5. Click on the **Add Rule** button, and choose 'All TCP' and under `Source` select 'Anywhere'. This allows people anywhere to connect to the database. Strictly speaking we can probably just open up port 3306, but we'll keep things simple! Note that port 22 is open, which is how you will connect via SSH or PuTTy.



Type	Protocol	Port Range	Source	
SSH	TCP	22	Custom	0.0.0.0/0
All TCP	TCP	0 - 65535	Anywhere	0.0.0.0/0, ::/0

6. Click **Review and Launch**, and then **Launch**. The next dialog allows you to create a private key which you will use to connect.



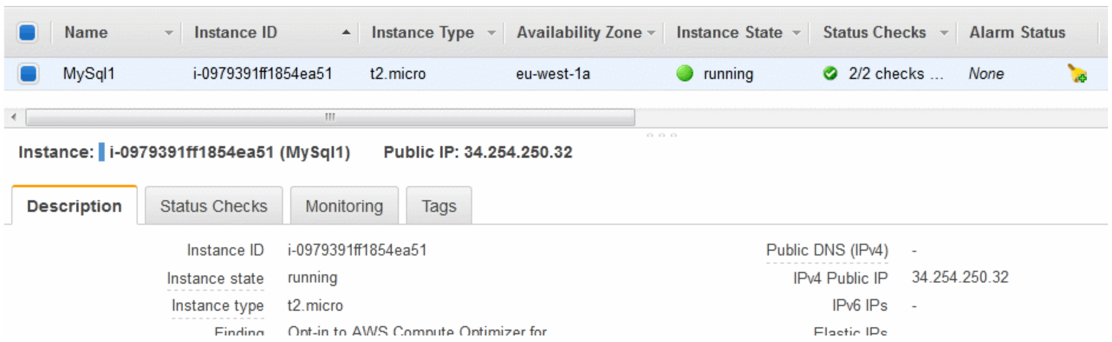
Create a new key pair

Key pair name

mysql1

Download Key Pair

7. Make sure you save the key to a place you can find it! If you lose it you will have to start all over again...! Click **Launch Instances**, and then **View Instances**.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
MySql1	i-0979391ff1854ea51	t2.micro	eu-west-1a	running	2/2 checks ...	None

Instance: i-0979391ff1854ea51 (MySql1)   Public IP: 34.254.250.32

Description | Status Checks | Monitoring | Tags

Instance ID	i-0979391ff1854ea51	Public DNS (IPv4)	-
Instance state	running	IPv4 Public IP	34.254.250.32
Instance type	t2.micro	IPv6 IPs	-
Finding	Opt-in to AWS Compute Optimizer for	Elastic IPs	

Note the Public IP. This is what you will use to connect to your machine.

8. If you are using Windows, we recommend downloading PuTTy to connect. You must use PuTTyGen to convert your private key file (`.pem`) to a PuTTy key file (`.ppk`). The key is the same, but the format of the file is slightly different. You then connect using PuTTy. The host will be:

```
ec2-user@34.254.250.32
```

or whatever your Public IP is.

9. If you are using Mac or Linux, open a terminal and connect via:

```
> ssh -i myslq1.pem ec2-user@34.254.250.32
```

10. If your key isn't liked, you may have to allow security on the `.pem` file itself, with:

```
> chmod 775 mysql1.pem
```

11. Once you have connected to your AWS instance, install Docker and start its service.

```
sudo bash
yum update -y
yum install docker -y
service docker start
```

12. Now run a Docker container, based on the MySql image, which will be fetched from DockerHub:

```
docker run --name mysql1 -p 3306:3306
     -e MYSQL_ROOT_PASSWORD=password -d mysql:5.5
```

This runs the Docker container containing a ready MySql instance, setting the password as '`password`' and calls the container '`mysql1`'.

13. Check out what has happened with:

    ```
    docker ps
    docker images
    ```

14. Now attach to your container with:

    ```
    docker exec -it mysql1 mysql -u root -p
    ```

    and on the next line type in the password `password`.

15. This attaches to the Docker container and will open a `mysql>` prompt, at which you type the following lines to create the database, a table and some data:

```
create database employeedb;
use employeedb;
create table employees (id int,
     firstname varchar(40), lastname varchar(40), age int );
insert into employees values (1,'Mickey','Mouse',40);
select * from employees;
\quit
```

16. This returns you to your terminal and your set-up is complete. The JDBC URL for the database will be:

    ```
    jdbc:mysql://xx.xx.xx.xx:3306/employeedb
    ```

    Where the X’s refer to your AWS instances Public IP address.