Completing the day1+2 exercises and hosting on DigitalOcean

Part-1: Complete and deploy the Quote exercise (part 1+2)

Complete the Quote exercise (ex-1, no-database) from day1+2, both the part implementing the user-functionality + the part that handles error scenarios.

When completed deploy the Application to your (DigitalOcean) Tomcat Server, via the Web Application Manager page as described in the document "Tomcat on your server".

This should hopefully work without any problems, since there is no database involved. If however "things" does not work as expected use the log-files on your droplet to pinpoint the probem (as described in the section "Log-files and Console Output" in the document "Tomcat on your server".

Part-2 Complete and deploy the Person exercise (part 1+2)

Complete the Person exercise (ex-2, database) from day1+2, both the part implementing the user-functionality + the part that handles error scenarios.

When completed, follow the instructions in the document *MySql On Your Server.pdf* (in the folder "instructions and software") and install/setup MySQL on your Droplet.

JPA and MySQL on Linux

We suggest that you in most scenarios use JPA's ability to create tables from the Entity classes, using either the *create* or *drop-and-create* options in the persistence.xml (remember to upload a version without this option once your tables are created).

If you are on a Windows system there is one thing you have to be aware of. Most names used with MySQL (table, column etc.- names) are not case-sensitive on Windows. This means that if you have an Entity class called `Person` it will map to a table called `person` on Windows. On Linux however, the same entity class will generate a table called `PERSON`.

The effect of this are, that if you are running a script via the persistence.xml file cased like below:

`Insert into person(firstName, lastName, phone)` it will work on windows, but fail on Linux, since the table name is PERSON and column names are FIRSTNAME, LASTNAME and PHONE.

The solution is; either to always write your scripts like:

```
insert into PERSON(FIRSTNAME,LASTNAME,PHONE) values ('Kurt','Wonnegut','123456');
```

or specifically name your *tables* and *columns* via JPA annotations like:

```
@Entity
@Table(name = "person") //This will map to a table person, also on LINUX
public class Person implements
```

Steps to prepare your first JPA/DigitalOcean project for deployment

This assumes you have installed MySQL as explained in the document *MySql On Your Server.pdf*

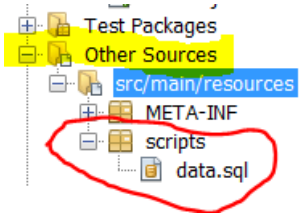
1) Create a new user ***tomcat*** as explained in the section "*Create a new user and grant privileges*". Use this user for all your tomcat/MySQL applications from now on (don't use this strategy for a production system).

2) On DigitalOcean connect to the MySQL console and create a new database ***person*** for this exercise as outlined below:

1. `mysql -u root -p` (you now have the mysql> prompt)
2. Write: `create database` **`person;`**
3. Verify it was created by writing: `show databases`

3) If not already done, create a script in your project as explained below:

Add a new folder **scripts** to "src/main/resources" as sketched to the right, and add a file **data.sql** to the folder.



Insert a number of persons via the script (observe the casing).

```
insert into PERSON values (null,'Kurt','Wonnegut','123456');
insert into PERSON values (null,Jane,'Wonnegut','795843');
insert into PERSON values (null,Peter,'Hansen','78964456');
```

Add this line to the properties section of your persistence.xml to run the script when using create or drop-and-create.

```
<property name="javax.persistence.sql-load-script-source" value="scripts/data.sql"/>
```

4) In your persistence.xml file, copy the existing persistence unit into the clipboard, and paste it back into the file as a new one with the same name. There are many strategies to handle different persistence-units. Here we will have two (one for local development and one for DigitalOcean) and comment out the one we don't need.

Change the values below to reflect the things we did on the remote MySQL (created a database *person*, created a user *tomcat* and a *password* for that user)

```
<!-- <persistence-unit name="pu" transaction-type="RESOURCE_LOCAL">    ……ORIGINAL PU……
</persistence-unit>-->
<persistence-unit name="pu" transaction-type="RESOURCE_LOCAL">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <class>entiny.Person</class>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/person"/>
    <property name="javax.persistence.jdbc.user" value="tomcat"/>
    <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
    <property name="javax.persistence.jdbc.password" value="INSERT_YOUR_OWN_PASSWORD"/>
    <property name="javax.persistence.schema-generation.database.action" value="create"/>
    <property name="javax.persistence.sql-load-script-source" value="scripts/data.sql"/>
  </properties>
</persistence-unit>
```

Recompile the project with this new persistence-unit (remember to comment out the original) and deploy the WAR-file via the Web Application Manager as for the Quote exercise above.