Integrating Spring Boot with MySQL

Introduction

For this course we will be using MySQL as the database for permanent data storage. We will use Java Persistence API (JPA) as an Object Relation Map (ORM) to map Java classes to SQL tables and Java object instances to records. These instructions are also documented in the following online slides (https://goo.gl /QFmEsk) and available online (https://goo.gl/i6X4ss). This assignment assumes the successful completion of an earlier assignment.

Learning Objectives

By the end of this assignment you should be able to

- Create and configure a remote MySQL database on AWS
- Connect to a remote MySQL database with MySQL Workbench
- Integrate a Spring Boot Web project with a remote MySQL database
- Map Java POJOs to a relational schema using Java Persistence API
- Expose a data model through a Web service endpoint

Creating a Remote MySQL Instance on Heroku

This section describes creating a remote MySQL database instance running on Heroku. These steps assume you have successfully deployed a Spring Boot application on Heroku. If not, make sure to follow the instructions for setting up a remote account, remote application, and a development environment as described in Settingup a Development Environment lecture. Once the environment is setup, create and deploy a Spring Boot application on Heroku as describe in Deploying Spring Boot Web Applications to Heroku. Finally, add a remote MySQL database to the remote development environment as described in Adding a Remote MySQL Database to a Spring Boot Web Application on Heroku. The general steps for setting up the environment on Heroku are listed below. Refer to the original documents for more details.

1. Install JDK 8 or later
2. Install Apache Maven
3. Install the Spring Boot framework
4. Install MySQL Workbench or some other MySQL client
5. Create an account on Heroku
6. Install the Heroku CLI
7. Create a simple Spring Boot Web application, e.g.,
   spring init --dependencies=web myapp

8. Deploy the Spring Boot application to Heroku, e.g.,
   heroku create

9. Add a MySQL remote database to the remote Spring Boot application on Heroku
10. Connect your local MySQL Workbench to the remote MySQL on Heroku

The connection string will contain the username, password, host and port. Make a note of this since you'll need it in later steps of this guide.

Creating a Remote MySQL Instance on AWS

This section describes creating a remote MySQL database instance running on AWS.

1. Login to the Amazon AWS console and expand *All Services*.
2. Under the *Database* section, select *RDS*.
3. In the *Amazon RDS* landing page, select *Launch* or *Get Started Now* to add a new RDS instance.
4. In the *Select engine* screen, select *MySQL* and then click *Next*.
5. In the *Choose use case* screen, select *Dev/Test - MySQL* and then click *Next*.
6. In the *Specify DB details* screen, keep the default settings, and choose the following configuration and then click *Next*. Use your own identifier, username, and password. The usernames, names, and identifiers shown in this document are based on a particular course, e.g., `cs5200`, semeter, e.g., `fall2018`, and your lastname, e.g., `annunziato`. Please use your particular values where applicable.

   a. DB instance class: `db.t2.micro`
   b. DB instance identifier: `cs5200-fall2018-annunziato`
   c. Master username: `jannunzi`
   d. Master password: `yourPa$$word123`
   e. Confirm password: `yourPa$$word123`

   Make note of the actual values used above since they will be used in later steps.

7. In the *Configure advanced settings* screen, select *Yes* for the *Public accessibility*. Also, keep the default settings, but choose the name of the database, e.g., `cs5200_fall2018_annunziato`. Note the use of underscores. This will be the name of the schema where tables and their records will be stored. Click on *Launch DB Instance* to continue. The database will take a few moments to be created after which you can navigate to it by clicking on *View DB Instance Details* or clicking *Instances* on the left.

8. The details screen will be titled with the DB instance identifier chosen earlier, e.g., `cs5200-fall2018-annunziato`. Scroll down to the *Connect* section. Make a note of

the *Endpoint* since it will be used later to connect remotely to the database. The endpoint is the name of the server machine where the database is running, e.g.,

```
cs5200-fall2018-annunziato.cne500ro4imj.us-west-2.rds.amazonaws.com
```

Also note the *Port* where the server is listening for incoming connections, e.g., `3306`. If the *Endpoint* is not yet available, wait a few more minutes while the database service is setup.

9. Note that the connection might not be publicly available by default as denoted by the configuration *Publicly accessible: No*. To make the connection publicly available, under the *Security group*, click on the *Inbound* security group. In the security group screen, click the *Inbound* tab, and then the *Edit* button. Under the *Source* column, select *Anywhere* from the dropdown, and click *Save*. Verify that the *Publicly accessible* setting is set to *Yes*.

Install, Configure, and Start MySQL Workbench

MySQL Workbench (Workbench) is a desktop application that provides a graphical user interface to easily interact with a MySQL database instance. Workbench will be used throughout to interact with MySQL running remotely on AWS or Heroku. To install Workbench, navigate to

https://dev.mysql.com/downloads/workbench/

and download Workbench for your particular operating system. Optionally signup and login, or scroll down a bit further and just start the download. After downloading, start the installation and accept all the defaults, and follow the steps below:

1. After installing, run the Workbench and create a new connection by clicking on the plus icon next to the *MySQL Connections* title.

2. In the *Setup New Connection* dialog, name your connection, e.g., `cs5200-connection`, and use the server name, port, master username, and master password noted earlier for the *Hostname*, *Port*, *Username*, and *Password* fields, and click *Ok*.

3. Click on the new connection to connect to the remote MySQL database and start interacting with it through Workbench.

Create a Simple, Sample Database

This section describes creating and interacting with a simple table with some sample data. From Workbench, create a new schema name based on the course you are enrolled in, e.g., `cs5200`.

To create a new schema, click on the *New Schema* button (shown at right). In the *new_schema* screen, type the name of the schema in the *Schema Name* field and click *Apply*. Verify that the following command will be executed and click *Apply* again and then *Close* (or *Finish*).

```
CREATE SCHEMA cs5200;
```

Make the new schema the default schema by double clicking it or right clicking it, and then selecting *Set as default schema*.

Create a table called `sample` with a single field called `message`. On Workbench, on the left, select the schema you created previously, and then click on the *New Table* icon to create a new table (shown at right). Type `sample` for the *Name*. Under the *Column* column, type *id*, select checkboxes PK and AI, and press enter to create a primary key named *id*.

Under the *id* field, type `message`, select *VARCHAR(45)* for the *Datatype*, and click *Apply*. Verify the following command will be executed and click *Apply* again and then *Close*.

```
CREATE TABLE cs5200.sample (
  id INT NOT NULL AUTO_INCREMENT,
  message VARCHAR(45) NULL,
  PRIMARY KEY (id));
```

On the left expand the *Tables*, expand the *sample* table, expand the *Columns*, and verify the following fields are present: *id* and *message*. Insert some sample data by right clicking the *sample* table, select *Send to SQL Editor*, and then *Insert Statement*. A new SQL editor window opens with the following SQL tempalate statement

```
INSERT INTO cs5200.sample
(id,
message)
VALUES
(<{id: }>,
<{message: }>);
```

Replace the following placeholders with the corresponding content

Placeholder	Replace with

`<{id: }>`	`1`
`<{message: }>`	`'Hello Jose Annunziato'`

Use YOUR NAME instead. To execute the SQL Insert command, click on the *Execute* button (shown on right). To verify that the content was saved, right click on the `sample` table again and select *Select Rows - Limit 1000*. A new SQL editor window appears with the following SQL statement

```
SELECT * FROM cs5200.sample;
```

Verify that the results are listed below the command window similar to the following

id	message
1	Hello Jose Annunziato

Replace the values in the *Insert* command to insert some more records. Use the following values:

id	message
2	SQL is great
3	Java is awesome

Insert the new records and verify the new records exist.

Integrate Spring Boot with MySQL

Start *Spring Tool Suite* (STS) and open the project created in an earlier assignment. Open the file `application.properties` under `src/main/resources` and type the following content to configure the datasource JPA will use to connect to the database

```
spring.datasource.url=jdbc:mysql://SERVER/SCHEMA
spring.datasource.username=USERNAME
spring.datasource.password=PASSWORD
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto=create
```

```
spring.jpa.show-sql=true
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
```

where `SERVER`, `SCHEMA`, `USERNAME`, and `PASSWORD` above, are the server name, master username, and master password noted earlier when setting up the aws server instance. Based on the values from earlier, the three lines would be

```
spring.datasource.url=jdbc:mysql://cs5200-fall2018-annunziato.cne500r
o4imj.us-west-2.rds.amazonaws.com/cs5200_fall2018_annunziato
spring.datasource.username=jannunzi
spring.datasource.password=myPa$$word123
```

Configure the project to download and install the MySQL Java connector library. Open the `pom.xml` file at the root of the project and type the following to download and install the Java MySQL library

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.45</version>
</dependency>
```

If you are using Java 9, you'll need to add the following `dependencies` section:

```
<dependency>
     <groupId>javax.xml.bind</groupId>
     <artifactId>jaxb-api</artifactId>
     <version>2.3.0</version>
</dependency>
```

If you are using Java 9, you'll need to add the following in the `plugins` section:

```
<plugin>
	<groupId>org.apache.maven.plugins</groupId>
	<artifactId>maven-war-plugin</artifactId>
	<version>3.2.0</version>
</plugin>
```

Configure a POJO to map to an SQL table and record instances. Open the `HelloObject.java` file you created in an earlier assignment and add the following content

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity(name="hello")
public class HelloObject {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
}
```

The `@Entity` annotation above maps the `HelloObject` class to a table called `hello`. All the properties in the class are mapped to fields of the same name. The `@Id` annotation configures the `id` property as the primary key. The `@GeneratedValue` annotation configures the `id` property to be generated automatically by the database, e.g., `AUTO_INCREMENT`. Spring Boot allows interacting with a database using the Java Persitence API (JPA) implemented through Spring's `JpaRepository` class. JPA allows interacting with a database saving and retrieving Java object instances, each representing records in a table in the database. To use JPA you need to create your own class that extends JpaRepository and configures the Java class and its primary key type. Create a `HelloRepository.java` interface with the following content

```
package edu.neu.cs5200.controllers.hello;

import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface HelloRepository
      extends JpaRepository<HelloObject, Integer> {
}
```

Once the `JpaRepository` has been declared, it can be used to save and retrieve records from the database. Add the following to the controller `HelloController.java` created in an earlier assignment:

```
import org.springframework.beans.factory.annotation.Autowired;

@RestController
public class HelloController {

      @Autowired
      HelloRepository helloRepository;

      @RequestMapping("/api/hello/insert")
      public HelloObject insertHelloObject() {
            HelloObject obj = new HelloObject("Hello Jose Annunziato!");
            helloRepository.save(obj);
            return obj;
      }
}
```

Replace Jose Annunziato with YOUR NAME. The above source creates a new instance of the `HelloObject` with the `message` property set to the constant literal `"Hello Jose Annunziato!"` and then saves it to the database. In addition, let's also create an endpoint that allows passing an arbitrary message as a parameter and then save it to the database. Add the following method to the `HelloController` class:

```
@RequestMapping("/api/hello/insert/{msg}")
public HelloObject insertMessage(@PathVariable("msg") String message)
{
      HelloObject obj = new HelloObject(message);
      helloRepository.save(obj);
      return obj;
}
```

The above code maps the URL pattern `/api/hello/insert/{msg}` to the `insertMessage()` method. The URL contains path variable `{msg}`, a placeholder that can be any string. The actual string value is then mapped to method parameter `message` using the `@PathVariable` annotation. Finally, let's add a method to retrieve all the records from the database and return them as a `List` of `HelloObject` instances using the `findAll()` method. Add the following to method to the `HelloController` class:

```
@RequestMapping("/api/hello/select/all")
public List<HelloObject> selectAllHelloObjects() {
	List<HelloObject> hellos =
		(List<HelloObject>)helloRepository.findAll();
	return hellos;
}
```

The above code maps the URL pattern `/api/hello/select/all` to the `selectAllHelloObjects()` method. The method uses the `helloRepository.findAll()` method to retrieve all records from the hello table and converts them to a `List` of `HelloObject` instances.

Enable remote REST APIs on AWS by extending `SpringBootServletInitializer` in your `SpringBootApplication` class you created in an earlier assignment, e.g., `Cs5200Fall2018AnnunziatoApplication`:

```
package edu.neu.cs5200;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;

@SpringBootApplication
public class Cs5200Fall2018AnnunziatoApplication
	extends SpringBootServletInitializer {

	@Override
	protected SpringApplicationBuilder
		configure(SpringApplicationBuilder application) {
		return application.sources(
			CS5200Fall2018AnnunziatoApplication.class);
	}
```

```
	public static void main(String[] args) {
		SpringApplication.run(
			Cs5200Fall2018AnnunziatoApplication.class, args);
	}
}
```

Save all files and rebuild the project. Right click on the project and select *Maven* and then *Update Project*. From the command line, navigate to the project directory and build the project using maven

```
mvn clean install
```

To test your new Webservice end point, start the Tomcat server and point your browser to

http://localhost:8080/api/hello/insert

Verify the server responds with the following JSON:

```
{
  "id": 1,
  "message": "Hello Jose Annunziato!"
}
```

Reload a couple of time and verify the `id` increments everytime you reload. List all the records by pointing your browser to http://localhost:8080/api/hello/select/all. Verify the server responds with the following JSON:

```
[
	{
		"id": 1,
		"message": "Hello Jose Annunziato!"
	},
	{
		"id": 2,
		"message": "Hello Jose Annunziato!"
	},
	{
		"id": 3,
		"message": "Hello Jose Annunziato!"
	},
```

```
]
```

Now try inserting some random messages, e.g., "JPA Rocks", "Spring's the Best", using the `insert/{msg}` URL pattern. Point your browser to the following URLs:

http://localhost:8080/api/hello/insert/JPA Rocks
http://localhost:8080/api/hello/insert/Spring's the Best

List all the records by pointing your browser to http://localhost:8080/api/hello/select/all. Verify the server responds with the following JSON:

```
[
	{
		"id": 1,
		"message": "Hello Jose Annunziato!"
	},
	{
		"id": 2,
		"message": "Hello Jose Annunziato!"
	},
	{
		"id": 3,
		"message": "Hello Jose Annunziato!"
	},
	{
		"id": 4,
		"message": "JPA Rocks"
	},
	{
		"id": 5,
		"message": "Spring's the Best"
	},
]
```

In Workbench, verify the data has been saving to the database by right clicking on the *hello* table on the left, and selecting *Select Rows - Limit 1000*. A new SQL editor window appears with the following SQL statement

```
SELECT * FROM cs5200.hello;
```

Verify that the results are listed below the command window similar to the following:

id	message
1	`Hello Jose Annunziato!`
2	`Hello Jose Annunziato!`
3	`Hello Jose Annunziato!`
4	`JPA Rocks`
5	`Spring's the Best`

Where the messages should include YOUR NAME instead. Redeploy the war file to the remote Web application deployed on AWS and verify the links work there as well. The links should look similiar to the following:

http://cs5200-fall2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/insert
http://cs5200-fall2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/insert/Some parameterized message
http://cs5200-fall2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/select/all

Note that when the application recompiles and/or redeploys, the tables are recreated and previous data is lost. This is fine for now while we are getting started in the early stages of development and we have not settled on a data model and the schema needs to reflect changes in the data model. When we settle on a data model, in later assignments, we will reconfigure JPA to not recreate the tables and loose our data.

Deliverables

As a deliverable add a section called "Integrating Spring Boot with MySQL" to your `Submission.md` file that contains three links to the Web service endpoints created in this assignment. The links should return JSON data similar in structure and content as verified throughout the assignment. The `Submission.md` should look as follows:

> ## Integrating Spring Boot with MySQL
>
> Insert a static hello message
> Insert a parameterized hello message
> Retrieve all hello messages

Add, commit and push all the work completed under the `Project-2` directory, to your course repository setup in an earlier assignment. Use the following message for your commit: `"Integrating Spring Boot with MySQL"`.