**Spring Data JPA with Spring Boot, Hibernate - Exercise**

**Hands on 1**

**Spring Data JPA - Quick Example**

Create a Eclipse Project using Spring Initializr · Go to https://start.spring.io/

· Change Group as “com.cognizant”

· Change Artifact Id as “orm-learn”

· In Options > Description enter "Demo project for Spring Data JPA and Hibernate"

· Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"

· Click Generate and download the project as zip

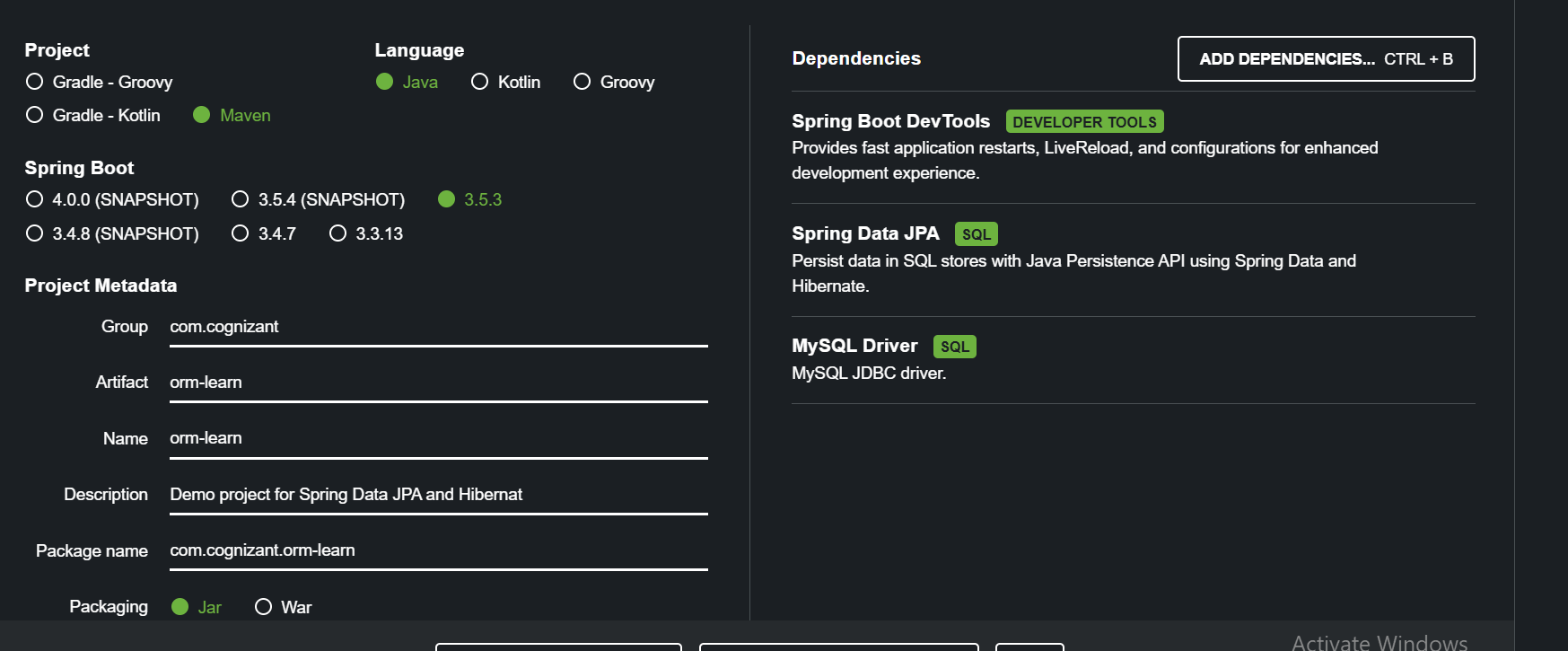
· Extract the zip in root folder to Eclipse Workspace

· Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"

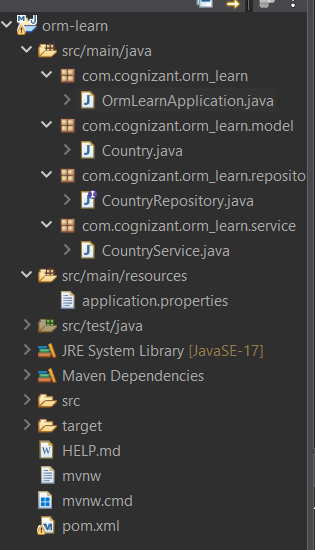
· Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

**=**

Spring project is created and the zip file is made

****

**Next we import it n our eclipse workplace**

****

**Next, we created the schema**

create schema ormlearn;

USE ormlearn;

**In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.**

**=**

**/orm-learn/src/main/resources/application.properties**

spring.application.name=orm-learn

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=debug

logging.level.org.hibernate.type.descriptor.sql=trace

# Console log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# MySQL Database config

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=Swarnadri@123

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**Include logs for verifying if main() method is called.**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

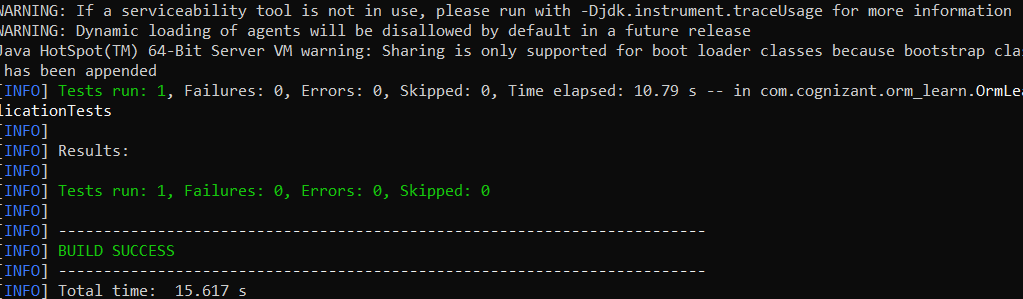
SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

}

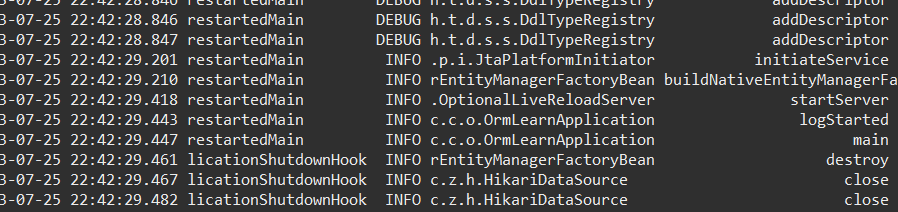
**Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line**

**=**

****

**Execute the OrmLearnApplication and check in log if main method is called.**

**=** we can see main method is called

****

**Country table creation**

**· Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.**

**Insert couple of records into the table**

**=**

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

DROP TABLE IF EXISTS country;

CREATE TABLE country (

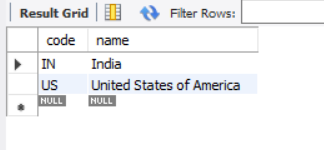
code VARCHAR(2) PRIMARY KEY,

name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');



**Persistence Class - com.cognizant.orm-learn.model.Country**

**· Open Eclipse with orm-learn project**

**· Create new package com.cognizant.orm-learn.model**

**· Create Country.java, then generate getters, setters and toString() methods.**

**· Include @Entity and @Table at class level**

**· Include @Column annotations in each getter method specifying the column name**.

=

package com.cognizant.orm\_learn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "name")

private String name;

// Getters and Setters

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// toString()

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

We have used Jakarta instead of javax as I am using spring 3.5.3 and javax is giving me errors

**Repository Class - com.cognizant.orm-learn.CountryRepository**

**· Create new package com.cognizant.orm-learn.repository**

**· Create new interface named CountryRepository that extends JpaRepository<Country, String>**

**· Define @Repository annotation at class level**

=

package com.cognizant.orm\_learn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.orm\_learn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Service Class - com.cognizant.orm-learn.service.CountryService**

**· Create new package com.cognizant.orm-learn.service**

**· Create new class CountryService**

**· Include @Service annotation at class level**

**· Autowire CountryRepository in CountryService**

**· Include new method getAllCountries() method that returns a list of countries.**

**· Include @Transactional annotation for this method**

**· In getAllCountries() method invoke countryRepository.findAll() method and return the result**

=

package com.cognizant.orm\_learn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Testing in OrmLearnApplication.java**

**· Include a static reference to CountryService in OrmLearnApplication class**

**Define a test method to get all countries from service.**

**Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.**

**=**

package com.cognizant.orm\_learn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.service.CountryService;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

List<Country> countries = countryService.getAllCountries();

LOGGER.debug("countries={}", countries);

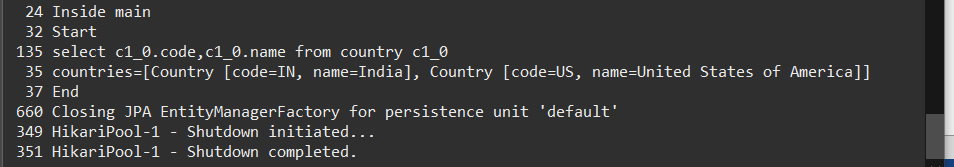
LOGGER.info("End");

}

}

**Execute main method to check if data from ormlearn database is retrieved.**

**=**

****

**We can see that inside main is debugged and cpunty code and name as output can be observed. So retrieval is working.**

**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA**

**ANS:**

**Java Persistence API (JPA)**

· JSR 338 Specification for persisting, reading and managing data from Java objects

|  |
| --- |
| **Type:** |

|  |
| --- |
| **Interface/contract** only – no implementation |

It defines that how Java objects map to database tables and how to manage entity life cycles

· Does not contain concrete implementation of the specification

· Hibernate is one of the implementation of JPA

**Hibernate**

· ORM Tool that implements JPA

Main idea includes Converting Java objects into database tables and vice versa

As it has its own native api, it can be used **with or without** JPA

**Spring Data JPA**

· Does not have JPA implementation, but reduces boiler plate code

· This is another level of abstraction over JPA implementation provider like Hibernate

Eliminates boilerplate code like findAll(), save(), deleteById() etc

· Manages transactions

|  |  |  |
| --- | --- | --- |
| **JPA** | **Hibernate** | **Spring data jpa** |
| Specification | Implementation | Abstraction over jpa and spring |
| It needs implementation | It does not need Implementation | Implementation is observed |
| Boilerplate is not reduced | Boilerplate is less | Boilerplate is highly reduced |
| Whatever easy to use | Kind of easy to use | Very easy to use |

**Code snippets comparison( as provided in the document)**

**Hibernate**

/\* Method to CREATE an employee in the database \*/

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA**

**EmployeeRespository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Comparison of example other than the provided ones**

**JPA:**

@Entity

public class Country {

@Id

private String code\_number;

private String name;

}

Logic is to be written too but here we show that it works through specification. We have to handle transactions and handle objects manually

**Hibernate:**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Book book = new Book();

book.setId(2L);

book.setTitle("Lets work with hibernate");

session.save(book);

tx.commit();

session.close();

Hibernate helps more directly but still writing all the steps in needed.

**Spring Data Jpa:**

@Entity

public class Book {

@Id

private Long id;

private String title;

}

@Repository

public interface BookRepository extends JpaRepository<Book, Long> { }

// In service or main method

@Autowired

private BookRepository bookRepo;

bookRepo.save(new Book(3L, "we are using spring data jpa"));