Spring Data JPA - Quick Example

Software Pre-requisites

- MySQL Server 8.0
- MySQL Workbench 8
- Eclipse IDE for Enterprise Java Developers 2019-03 R
- Maven 3.6.2

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.

> mysql -u root -p

mysql> create schema ormlearn;

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

 # Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

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

Log pattern

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

Database configuration

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=root

Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

- 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
- 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");
}
```

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

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. OrmLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
    1. Walkthrough all the configuration defined in XML file
    2. Open 'Dependency Hierarchy' and show the dependency tree.

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.

```
create table country(co_code varchar(2) primary key, co_name
varchar(50));
```

- Insert couple of records into the table

```
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.

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.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

    // toString()
```

}

Notes:

- @Entity is an indicator to Spring Data JPA that it is an entity class for the application
- @Table helps in defining the mapping database table
- @Id helps is defining the primary key
- @Column helps in defining the mapping table column

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

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

import org.springframework.stereotype.Repository;


import com.cognizant.ormlearn.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

Testing in OrmLearnApplication.java

- Include a static reference to CountryService in OrmLearnApplication class

```
private static CountryService countryService;
```

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

```
private static void testGetAllCountries() {
    LOGGER.info("Start");
    List<Country> countries = countryService.getAllCountries();
    LOGGER.debug("countries={}", countries);
    LOGGER.info("End");
}
```

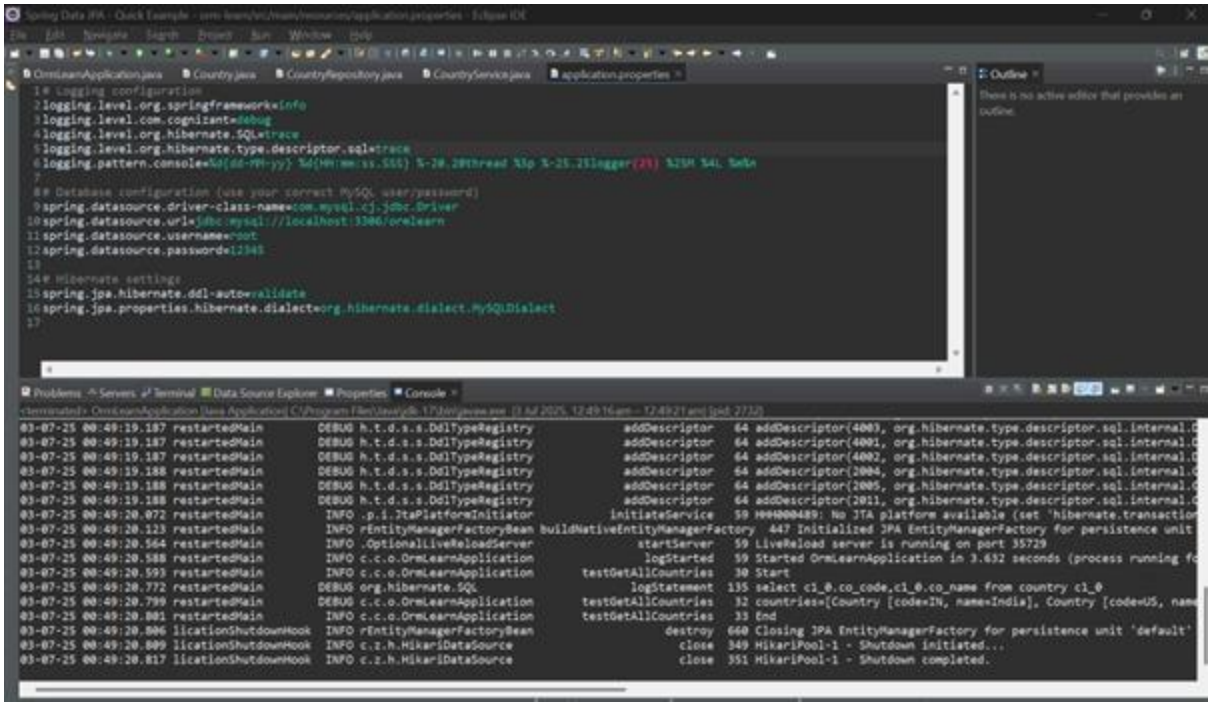
- Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

```
    ApplicationContext context =
SpringApplication.run(OrmLearnApplication.class, args);
    countryService = context.getBean(CountryService.class);


    testGetAllCountries();
```

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

Out put:


Spring Data JPA - Quick Example - com-learn/src/main/resources/application.properties - Eclipse IDE
File Edit Navigate Search Project Run Window Help
OrmLearnApplication.java
Country.java
CountryRepository.java
CountryService.java
application.properties
# Logging configuration
logging.level.org.springframework=info
logging.level.com.cognizant=debug
logging.level.org.hibernate.SQL=trace
logging.level.org.hibernate.type.descriptor.sql=trace
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n
# Database configuration (use your correct MySQL user/password)
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=1234$
# Hibernate settings
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
Outline
There is no active editor that provides an outline.
Problems Servers Terminal Data Source Explorer Properties Console
03-07-25 00:49:19.187 restartedMain DEBUG h.t.d.s.s.DdlTypeRegistry addDescriptor 64 addDescriptor(4001, org.hibernate.type.descriptor.sql.internal.
03-07-25 00:49:19.187 restartedMain DEBUG h.t.d.s.s.DdlTypeRegistry addDescriptor 64 addDescriptor(4002, org.hibernate.type.descriptor.sql.internal.
03-07-25 00:49:19.188 restartedMain DEBUG h.t.d.s.s.DdlTypeRegistry addDescriptor 64 addDescriptor(2004, org.hibernate.type.descriptor.sql.internal.
03-07-25 00:49:19.188 restartedMain DEBUG h.t.d.s.s.DdlTypeRegistry addDescriptor 64 addDescriptor(2005, org.hibernate.type.descriptor.sql.internal.
03-07-25 00:49:19.188 restartedMain DEBUG h.t.d.s.s.DdlTypeRegistry addDescriptor 64 addDescriptor(2011, org.hibernate.type.descriptor.sql.internal.
03-07-25 00:49:20.072 restartedMain INFO .p.i.JtaPlatformInitiator initiateService 59 HHH000489: No JTA platform available (set 'hibernate.transactio
03-07-25 00:49:20.123 restartedMain INFO rEntityManagerFactoryBean buildNativeEntityManagerFactory 447 Initialized JPA EntityManagerFactory for persistence unit
03-07-25 00:49:20.564 restartedMain INFO .OptionalLiveReloadServer startServer 59 LiveReload server is running on port 35729
03-07-25 00:49:20.588 restartedMain INFO c.c.o.OrmLearnApplication logStarted 59 Started OrmLearnApplication in 3.632 seconds (process running f
03-07-25 00:49:20.593 restartedMain INFO c.c.o.OrmLearnApplication testGetAllCountries 30 Start
03-07-25 00:49:20.772 restartedMain DEBUG org.hibernate.SQL logStatement 135 select c1_0.co_code,c1_0.co_name from country c1_0
03-07-25 00:49:20.799 restartedMain DEBUG c.c.o.OrmLearnApplication testGetAllCountries 32 countries=[Country [code=IN, name=India], Country [code=US, nam
03-07-25 00:49:20.801 restartedMain INFO c.c.o.OrmLearnApplication testGetAllCountries 33 End
03-07-25 00:49:20.806 licationShutdownHook INFO rEntityManagerFactoryBean destroy 660 Closing JPA EntityManagerFactory for persistence unit 'default'
03-07-25 00:49:20.809 licationShutdownHook INFO c.z.h.HikariDataSource close 349 HikariPool-1 - Shutdown initiated...
03-07-25 00:49:20.817 licationShutdownHook INFO c.z.h.HikariDataSource close 351 HikariPool-1 - Shutdown completed.

Example 2: Difference between JPA, Hibernate and Spring Data JPA

- Explain the difference between Java Persistence API, Hibernate and Spring Data JPA
    - JPA (Java Persistence API), JPA is a specification (JSR 338), JPA does not have implementation, Hibernate is one of the implementation for JPA, Hibernate is a ORM tool, Spring Data JPA is an abstraction above Hibernate to remove boiler plate code when persisting data using Hibernate.
        - Difference between Spring Data JPA and Hibernate - https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1
        - Intro to JPA - https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

Process:

Step 1: Ensure You Have These Tools Installed

- Java 17 or 21+
- Maven or Gradle
- An IDE like IntelliJ or Eclipse
- Optional: Docker for database

Step 2: Sample `pom.xml` (if using Maven)

```
<dependencies>
    <!-- Spring Boot Starter Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- H2 Database for in-memory testing -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Spring Boot Starter Web (if REST APIs involved) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Optional: Lombok for cleaner code -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Step 3: Sample application.properties

spring.datasource.url=jdbc:h2:mem:testdb

```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# Optional for debugging
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Step 4: Sample JPA Entity

```
import jakarta.persistence.*;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    // Getters & Setters
}
```

Step 5: Sample Repository Interface

```
import org.springframework.data.jpa.repository.JpaRepository;
public interface UserRepository extends JpaRepository<User, Long> {
    User findByName(String name);
}
```

Step 6: Main Class

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApp {
    public static void main(String[] args) {
```

```
        SpringApplication.run(MyApp.class, args);
    }
}
```