**Hands on 1**

**Spring Data JPA - Quick Example**

**Solution:**

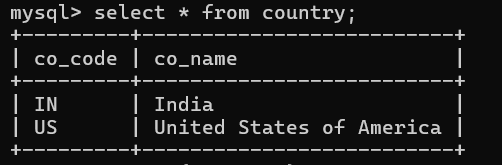
Database Creation: (ormlearn):

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

*insert into country values ('US', 'United States of America');*

*insert into country values ('US', 'United States of America');*

**Created Table:**

****

After the import of the Maven Project from “start.spring.io”

Country.java:

**package** com.cognizant.ormlearn.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 = "co\_code")

**private** String code;

@Column(name = "co\_name")

**private** String name;

**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; }

@Override

**public** String toString() {

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

}

}

CountryRepository.java:

**package** com.cognizant.ormlearn.repository;

**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> {

}

CountryService.java:

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** org.springframework.transaction.annotation.Transactional;

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

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

}

OrmLearnApplication.java:

**package** com.cognizant.ormlearn;

**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.ormlearn.model.Country;

**import** com.cognizant.ormlearn.service.CountryService;

**import** java.util.List;

@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);

*countryService* = context.getBean(CountryService.**class**);

*testGetAllCountries*();

}

**private** **static** **void** testGetAllCountries() {

***LOGGER***.info("Start");

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

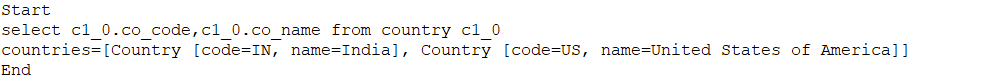
***LOGGER***.debug("countries={}", countries);

***LOGGER***.info("End");

}

}

The data are successfully fetched from the database.

OUTPUT:

**Hands on 4**

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

**Solution:**

**Java Persistence API (JPA)**

JPA (Java Persistence API) is a **specification (JSR 338)** that defines a standard for persisting, reading, and managing data from Java objects in a relational database. It provides **standard annotations** (such as @Entity, @Id) and APIs (such as EntityManager) to handle persistence but **does not provide a concrete implementation**. An implementation provider like Hibernate or EclipseLink is required to perform the actual ORM operations defined by JPA.

**Hibernate**

Hibernate is an **Object-Relational Mapping (ORM) framework** that **implements the JPA specification**. It is responsible for **mapping Java objects to database tables**, managing SQL generation, connection pooling, caching, and entity state management. Hibernate offers both **JPA APIs (EntityManager)** and its own **native APIs (Session, Transaction)**, allowing flexible and fine-grained control over database operations. Unlike JPA, Hibernate is a complete implementation that can be directly used for persistence operations in Java applications.

**Spring Data JPA**

Spring Data JPA is a **Spring module that provides an abstraction layer over JPA**, making data access simpler and reducing boilerplate code. It requires a **JPA implementation provider like Hibernate** to work under the hood. By using repository interfaces such as JpaRepository and CrudRepository, Spring Data JPA enables **automatic CRUD operations**, supports **automatic transaction management using @Transactional**, and provides query generation through method naming conventions and @Query annotations, allowing rapid and cleaner data access in Spring-based applications.

CODE COMPARISONS:

**Using Hibernate**

We handle **session management, transactions, and exception handling manually** while persisting an employee entity. In this approach, we open a session, begin a transaction, save the employee object, commit the transaction upon success, and handle rollbacks explicitly in case of exceptions.

**Using Spring Data JPA**

Spring Data JPA simplifies the persistence operation by **eliminating manual session and transaction management**.

In this approach, we define a repository interface extending JpaRepository, which provides built-in methods like save(). We inject this repository into the service class, and within a @Transactional method, we persist the employee entity using employeeRepository.save(employee), without managing transactions or sessions explicitly.

Through this comparison, we understand that JPA provides the **guidelines and standards** for persistence, Hibernate **implements these standards**, and Spring Data JPA **simplifies the use of these standards**, reducing the effort required to perform CRUD operations while maintaining clean and maintainable code in Spring-based applications.