# Spring Data JPA With Hibernate

**Hands On 1:Spring Data JPA - Quick Example** **Country.java** src/main/java/com/cognizant/ormlearn/model/Country.java

package com.cognizant.ormlearn.model; 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 = "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** src/main/java/com/cognizant/ormlearn/repository/CountryRepository.java package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Country;

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

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java** src/main/java/com/cognizant/ormlearn/service/CountryService.java package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() { return countryRepository.findAll();

}

}

**OrmLearnApplication.java** src/main/java/com/cognizant/ormlearn/OrmLearnApplication.java package com.cognizant.ormlearn;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService; 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 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");

}

}

# application.properties

src/main/resources/application.properties

# Spring log levels logging.level.org.springframework=info logging.level.com.cognizant=debug

# Hibernate logs logging.level.org.hibernate.SQL=trace logging.level.org.hibernate.type.descriptor.sql=trace

# Console log format

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread

%5p %-25.25logger{25} %25M %4L %m%n

# MySQL 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=\*\*\*\*\*

# Hibernate dialect and mode spring.jpa.hibernate.ddl-auto=validate

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

# SQL: Create & Insert in country Table

create table country (

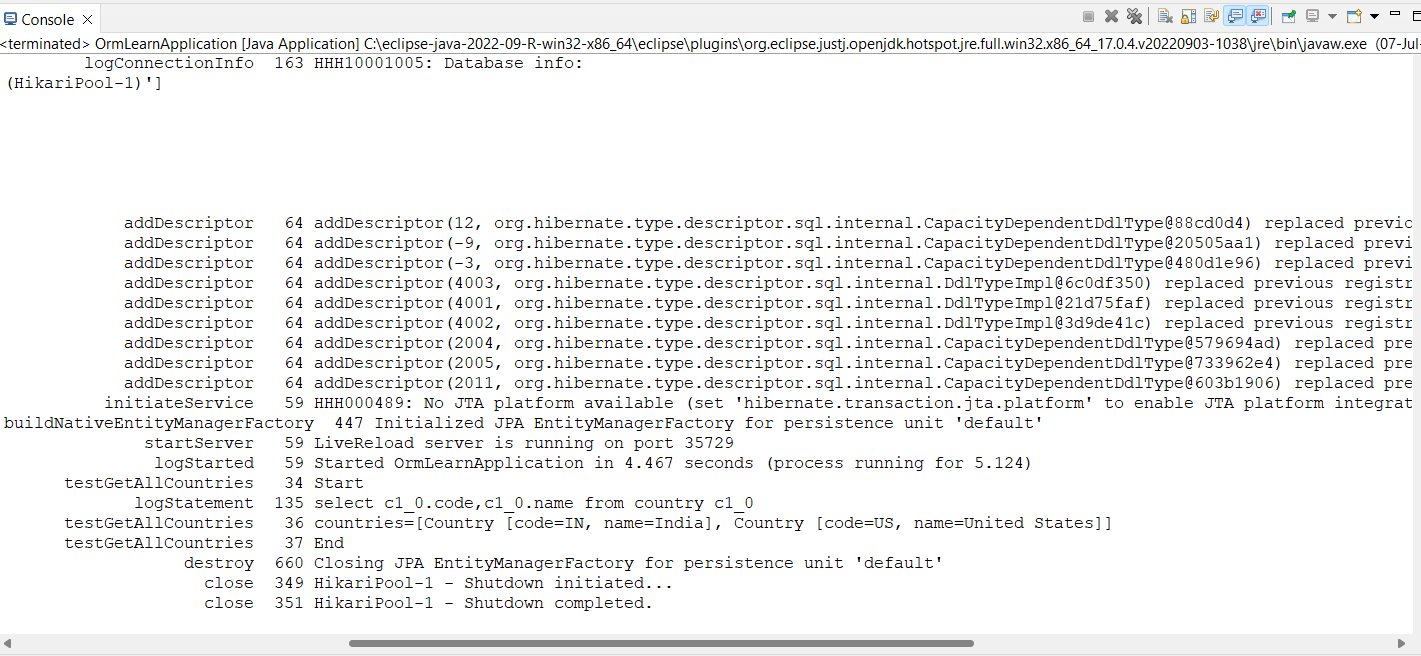
co\_code varchar(2) primary key, co\_name varchar(50)

);

insert into country values ('IN', 'India');

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

**OUTPUT**

****

**Hands On 4-Difference between JPA, Hibernate and Spring Data JPA**

**Java Persistence API (JPA)**

* JPA is a Java specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.
* It provides a standard API for ORM (Object Relational Mapping) in Java.
* JPA is only a set of interfaces; it does not provide implementation on its own.
* You need an implementation like Hibernate or EclipseLink to use JPA.
* JPA defines annotations like @Entity, @Table, @Id, etc., for mapping Java classes to database tables.

# Hibernate

* Hibernate is a concrete implementation of the JPA specification.
* It is an ORM tool that automatically handles object-database mapping.
* Hibernate includes features like lazy loading, caching, dirty checking, and automatic schema generation.
* It can be used with or without JPA.
* When used with JPA, Hibernate acts as the engine that performs the actual database operations.

# Spring Data JPA

* Spring Data JPA is a part of the Spring Framework that builds on top of JPA.
* It does not implement JPA, but simplifies JPA-based data access using repository abstraction.
* It removes boilerplate code for common CRUD operations.
* Spring Data JPA automatically implements repository interfaces based on naming conventions.
* It also manages transactions and integrates easily with Spring Boot and Spring configuration.
* Spring Data JPA usually works with Hibernate under the hood as the default JPA provider.

# Code Comparison

**Using Hibernate (Manual Approach):**

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;

}

This approach requires manually opening a session, starting a transaction, saving the entity, and handling rollback or commit.

# Using Spring Data JPA : EmployeeRepository.java:

public interface EmployeeRepository extends JpaRepository<Employee, Integer>

{

}

# EmployeeService.java:

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) { employeeRepository.save(employee);

}

In this approach, all the boilerplate code is removed. Spring handles the session, transaction, and persistence automatically.

# Summary

JPA is just a specification. Hibernate is one implementation of JPA that provides the actual functionality to perform ORM operations. Spring Data JPA is a convenient abstraction over JPA and Hibernate that makes development faster and cleaner by reducing repetitive code and providing ready-to-use repository interfaces.