# SKILL- Spring-Data-Jpa

**Spring Data JPA - Quick Example**

## Purpose:

To demonstrate how to integrate Spring Boot with MySQL using Spring Data JPA.  
It helps in understanding entity mapping, repository usage, and retrieving data from a relational database.  
Also teaches Maven setup, logging, and layered application structure.

# application.properties

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update  
logging.level.org.hibernate.SQL=DEBUG

# 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 com.cognizant.ormlearn.model.Country;  
  
public interface CountryRepository extends JpaRepository<Country, String> {  
  
}

# CountryService.java

package com.cognizant.ormlearn.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.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();  
 }  
}

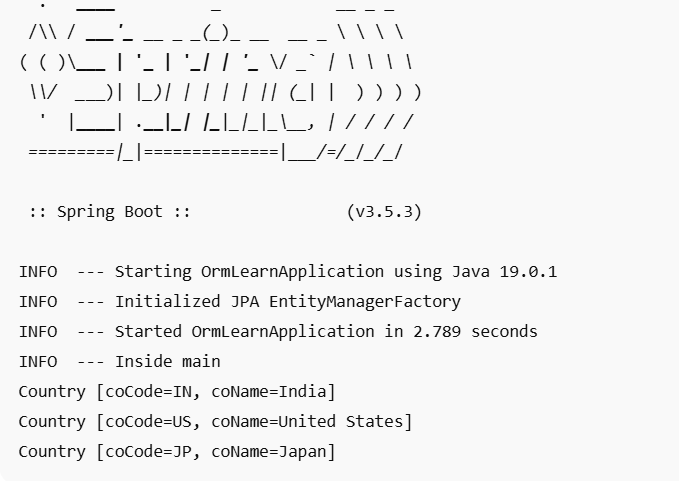
# OrmLearnApplication.java

package com.cognizant.orm\_learn;  
  
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 {  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 System.out.println("Inside main");  
 CountryService countryService = context.getBean(CountryService.class);  
 List<Country> countries = countryService.getAllCountries();  
 countries.forEach(System.out::println);  
 }  
}

# data.sql

INSERT INTO country (co\_code, co\_name) VALUES ('IN', 'India');  
INSERT INTO country (co\_code, co\_name) VALUES ('US', 'United States');  
INSERT INTO country (co\_code, co\_name) VALUES ('CN', 'China');

## Output :



# Difference between JPA, Hibernate and Spring Data JPA

## Purpose:

To explain the conceptual and practical differences between JPA, Hibernate, and Spring Data JPA.  
It highlights how each layer simplifies persistence—from specification (JPA), to implementation (Hibernate), to abstraction and reduced boilerplate (Spring Data JPA).

# Hibernate Example:

Hibernate is an ORM (Object-Relational Mapping) tool that implements the JPA specification.

hibernate.cfg.xml:  
<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">org.h2.Driver</property>

<property name="hibernate.connection.url">jdbc:h2:mem:testdb</property>

<property name="hibernate.connection.username">sa</property>

<property name="hibernate.connection.password"></property>

<property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<property name="show\_sql">true</property>

</session-factory>

</hibernate-configuration>

## Employee.java

package com.example.hibernate;

public class Employee {

private int id;

private String name;

public Employee() {}

public Employee(String name) {

this.name = name;

}

// Getters and setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + "]";

}

}

## HibernateUtil.java

package com.example.hibernate;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

private static final SessionFactory sessionFactory;

static {

try {

sessionFactory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Employee.class)

.buildSessionFactory();

} catch (Throwable ex) {

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

}

## Main.java

package com.example.hibernate;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class Main {

public static void main(String[] args) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee emp = new Employee("John Doe");

session.save(emp);

tx.commit();

System.out.println("Employee Saved: " + emp);

} catch (Exception e) {

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

e.printStackTrace();

} finally {

session.close();

}

}

}

## Output:

# 

# Spring Data JPA Example

Spring Data JPA reduces the boilerplate code needed to persist data.  
Here is an example using Spring Boot and Spring Data JPA.

## Employee.java

package com.example.springdatajpa.entity;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

public Employee() {}

public Employee(String name) { this.name = name; }

// Getters, setters, toString...

}

## EmployeeRepository.java

package com.example.springdatajpa.repository;

import com.example.springdatajpa.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}

## EmployeeService.java

package com.example.springdatajpa.service;

import com.example.springdatajpa.entity.Employee;

import com.example.springdatajpa.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository repo;

@Transactional

public void addEmployee(Employee emp) {

repo.save(emp);

}

}

**SpringDataJpaApplication.java**

package com.example.springdatajpa;

import com.example.springdatajpa.entity.Employee;

import com.example.springdatajpa.service.EmployeeService;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringDataJpaApplication implements CommandLineRunner {

@Autowired

private EmployeeService service;

public static void main(String[] args) {

SpringApplication.run(SpringDataJpaApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

Employee emp = new Employee("Jane Smith");

service.addEmployee(emp);

System.out.println("Employee saved: " + emp);

}

}

# application.properties

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create

spring.jpa.show-sql=true

## Output:



🔁 Code Comparison: Hibernate vs Spring Data JPA

| Aspect | Hibernate (Manual ORM) | Spring Data JPA (Simplified ORM) |
| --- | --- | --- |
| Configuration | hibernate.cfg.xml with all DB properties manually defined | application.properties with DB properties; Spring Boot auto-configures everything |
| Entity Definition | Employee class with getters/setters and no annotations shown in your sample | @Entity, @Id, @GeneratedValue annotations are used for mapping |
| Session Handling | Manually opening/closing Hibernate Session and Transaction | Handled automatically by Spring; uses @Transactional |
| DAO Layer | Main.java performs data access directly using Hibernate API (session.save(emp)) | EmployeeRepository extends JpaRepository and is autowired in EmployeeService |
| Transaction Management | Must explicitly begin/commit/rollback transactions | Use @Transactional annotation |
| Main Class / Runner | Main.java with SessionFactory usage | SpringDataJpaApplication.java implements CommandLineRunner and uses dependency injection |
| Boilerplate Code | More: manual session factory, session, transactions, exception handling | Less: handled by Spring Boot & Spring Data |