# Spring Data – JPA

**Objective**

# 1. Need and Benefit of ORM

**What is ORM?**  
ORM (Object-Relational Mapping) is a programming technique that allows developers to interact with a relational database using object-oriented paradigms.

**Need for ORM:**  
- Bridges the gap between object-oriented languages (like Java) and relational databases (like MySQL).  
- Manual conversion (Object ↔ Table) is time-consuming and error-prone.  
- Promotes abstraction and reusability in data access logic.

**Benefits of ORM:**  
- Reduces boilerplate code.  
- Improves developer productivity.  
- Ensures database independence.  
- Automatic SQL generation and object caching.  
- Supports transaction management.  
- Maintains relationships between objects (one-to-many, many-to-many, etc.).

# 2. Need and Benefit of Spring Data JPA

**Evolution of ORM:**  
- Initially used JDBC and Hibernate with XML configurations.  
- Then moved to Annotation-based Hibernate.  
- Now, Spring Data JPA provides more abstraction on top of Hibernate.

**Why Spring Data JPA?**  
- Reduces boilerplate DAO code.  
- Uses powerful repository interfaces like JpaRepository.  
- Supports method query derivation.  
- Easily integrates with databases like H2 and MySQL.  
- Supports pagination, sorting, custom queries.

**Benefits:**  
- Open-source and lightweight.  
- Eliminates need for explicit SQL.  
- Supports in-memory DB (for testing) and production DBs (like MySQL).  
- Seamless integration with Spring Boot.

# 3. Core Objects of Hibernate Framework

1. SessionFactory: Immutable and thread-safe. Created once and used to open Sessions. Represents the database configuration.

2. Session: Represents a single unit of work. Not thread-safe. Used to perform CRUD operations.

3. Transaction: Encapsulates a unit of work. ACID-compliant. Must begin and commit/rollback.

4. Connection Provider: Supplies JDBC connections to Hibernate. Abstracts the actual database connection pooling mechanism.

5. TransactionFactory: Produces instances of Transaction. Internally manages the transaction boundaries.

# 4. ORM Implementation with Hibernate

**XML Configuration:**  
- Persistence Class: POJO mapped to table.  
- Mapping File: .hbm.xml describes class-table mapping.  
- Config File: hibernate.cfg.xml includes DB credentials, dialect, etc.  
**Steps:**  
 1. Create mapping and config XMLs.  
 2. Load config via Configuration class.  
 3. Get SessionFactory, open Session.  
 4. Begin and commit transaction.

**Annotation Configuration:**  
- Use annotations like @Entity, @Table, @Id, @Column.  
- Load hibernate.cfg.xml, but mapping is via annotations.

# 5. Difference Between JPA, Hibernate, and Spring Data JPA

Comparison Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Type** | **Hibernate** | **Spring Data JPA** |
| **What it is** | A set of interfaces and annotations | ORM tool and JPA implementation | Framework that simplifies JPA-based persistence |
| **Boilerplate Code** | Requires writing queries and DAO | Reduces some boilerplate | Minimizes code further using repository pattern |
| **Ease of Use** | Medium | Easier than JPA | Easiest (automatic query generation) |
| **Standalone?** | Cannot be used alone (only API) | Can be used directly | Depends on JPA provider (like Hibernate) |
| **Annotations Used** | @Entity, @Id, @Table, etc. | Same as JPA (inherits from it) | Uses JPA annotations and Spring annotations |
| **Integration** | Needs provider like Hibernate | Direct integration with JDBC/DB | Integrates with Spring Boot |
| **Learning Curve** | Moderate | Easy | Very Easy |
| **Advanced Features** | Limited to specification | Provides lazy loading, caching, etc. | Supports CRUD, pagination, sorting, custom queries |
| **Customization** | Requires manual query definition | Custom queries supported | Method naming generates queries automatically |

# 6. DML Using Spring Data JPA

**Steps:**  
1. Add Dependencies: Spring Data JPA, Spring Boot Starter Data JPA, DB Driver (H2/MySQL).  
2. Entity Class: Use @Entity, @Id, @Column annotations.  
3. Repository:  
 - Create interface extending JpaRepository<Entity, ID>.  
 - Custom query: findByEmail(String email).

**Application Properties:**  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

**DML Operations:**  
- save(entity) → insert/update.  
- findById(id) → read.  
- deleteById(id) → delete.  
- Define query methods (e.g., findByName(String name)).

**Example:**  
public interface StudentRepository extends JpaRepository<Student, Long> {  
 List<Student> findByDepartment(String dept);  
}

**Hands-On 1 Spring Data JPA - Quick Example (Mandatory)**

**Creating Schema and Table**

CREATE DATABASE IF NOT EXISTS ormlearn;

USE ormlearn;

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');

INSERT INTO country VALUES ('JP', 'Japan');

INSERT INTO country VALUES ('FR', 'France');

INSERT INTO country VALUES ('AU', 'Australia');

**application.properties**

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

# MySQL DB 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=root

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.prope**rties.hibernate.dialect=org.hibernate.dialect.MySQLDialect**

**OrmLearnApplication.java**

package com.cognizant.ormlearn;

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.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.ApplicationContext;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication(scanBasePackages = "com.cognizant.ormlearn")

@EnableJpaRepositories("com.cognizant.ormlearn.repository")

@EntityScan("com.cognizant.ormlearn.model")

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);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

}

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.\*;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@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;

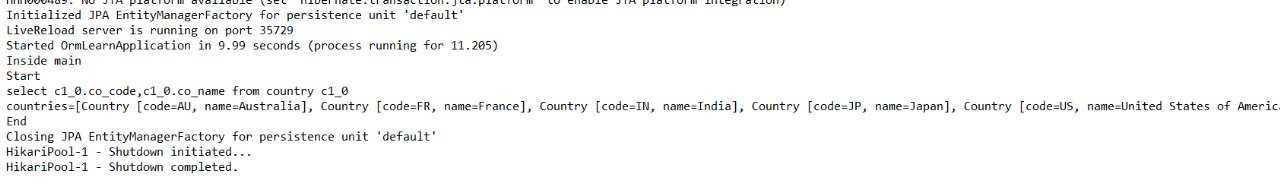
@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Output**

**Hands-on 4 Difference between JPA, Hibernate and Spring Data JPA (Mandatory)**

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

**1. JPA (Java Persistence API)**

* Definition: JPA is a Java specification (JSR 338) that defines a standard for object-relational mapping (ORM) between Java classes and relational databases.
* Nature: It is just a specification, meaning it provides guidelines but no actual code or implementation.
* Usage: It is used via an implementation provider such as Hibernate, EclipseLink, etc.
* Purpose: Helps to persist, retrieve, update, and delete Java objects in a relational database in a standardized way.

Example: You write @Entity, @Table, @Id, etc. based on JPA annotations, but they require an implementation like Hibernate underneath to work.

**2. Hibernate**

* Definition: Hibernate is an ORM (Object Relational Mapping) framework and the most popular implementation of JPA.
* Function: It provides actual code to perform database operations using objects.
* Features:
  + Handles connection, transactions, session management.
  + Converts Java classes to database tables and vice versa.
  + Provides features like lazy loading, caching, and HQL (Hibernate Query Language).
* How it relates to JPA: It implements the JPA interfaces and annotations, so when you use JPA with Hibernate, Hibernate handles the actual database interaction.

**3. Spring Data JPA**

* Definition: Spring Data JPA is not an implementation of JPA. It is a Spring module that provides an abstraction layer over JPA implementations like Hibernate.
* Goal: To reduce boilerplate code (e.g., writing queries, DAO classes).
* Key Benefits:
  + Built-in methods like findAll(), save(), deleteById() reduce the need to write SQL or HQL.
  + Allows creating queries using method names (e.g., findByName()).
  + Manages transactions automatically if @Transactional is used.
* Dependency: Still needs an actual JPA implementation underneath (like Hibernate).

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification (API) | ORM Framework | Spring abstraction over JPA |
| **Implementation** | No | Yes | No (uses JPA implementations) |
| **Reduces Boilerplate** | Partially | No | Yes |
| **Transaction Mgmt** | Needs manual config | Provided | Auto-managed |
| **Query Writing** | JPQL | HQL | Method name / JPQL |

**Code Comparison: Hibernate vs Spring Data JPA**

**Using Hibernate**

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;

}

**Disadvantages**:

* Manual session handling
* Explicit transaction control
* More boilerplate code

**Code**

**Hibernate.cfg.xml**

<?xml *version*="1.0" *encoding*="utf-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <property *name*="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

        <property *name*="hibernate.connection.url">jdbc:mysql://localhost:3306/orm\_learn</property>

        <property *name*="hibernate.connection.username">root</property>

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

        <property *name*="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

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

        <property *name*="hbm2ddl.auto">update</property>

        <property *name*="hibernate.current\_session\_context\_class">thread</property>

        <mapping *class*="com.example.hibernate.model.Employee" />

    </session-factory>

</hibernate-configuration>

**App.java**

package com.example.hibernate;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import com.example.hibernate.model.Employee;

public class App {

    public static void main(String[] args) {

        SessionFactory factory = new Configuration()

                .configure("hibernate.cfg.xml")

                .addAnnotatedClass(Employee.class)

                .buildSessionFactory();

        Session session = factory.getCurrentSession();

        try {

            Employee emp = new Employee();

            emp.setName("Elsa");

            session.beginTransaction();

            session.save(emp);

            session.getTransaction().commit();

            System.out.println("Saved Successfully!");

        } finally {

            factory.close();

        }

    }

}

**Employee.java**

package com.example.hibernate.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private int id;

    @Column(name = "name")

    private String name;

    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; }

}

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

                             http://maven.apache.org/maven-v4\_0\_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.hibernate</groupId>

  <artifactId>hibernate-demo</artifactId>

  <version>1.0-SNAPSHOT</version>

  <packaging>jar</packaging>

  <name>hibernate-demo</name>

  <url>http://maven.apache.org</url>

  <dependencies>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>3.8.1</version>

      <scope>test</scope>

    </dependency>

    <dependency>

      <groupId>org.hibernate</groupId>

      <artifactId>hibernate-core</artifactId>

      <version>6.3.1.Final</version>

    </dependency>

   <dependency>

    <groupId>com.mysql</groupId>

    <artifactId>mysql-connector-j</artifactId>

    <version>8.0.33</version>

</dependency>

    <dependency>

      <groupId>jakarta.persistence</groupId>

      <artifactId>jakarta.persistence-api</artifactId>

      <version>3.1.0</version>

    </dependency>

    <dependency>

      <groupId>org.slf4j</groupId>

      <artifactId>slf4j-api</artifactId>

      <version>2.0.9</version>

    </dependency>

    <dependency>

      <groupId>org.slf4j</groupId>

      <artifactId>slf4j-simple</artifactId>

      <version>2.0.9</version>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <groupId>org.codehaus.mojo</groupId>

        <artifactId>exec-maven-plugin</artifactId>

        <version>3.1.0</version>

        <configuration>

          <mainClass>com.example.hibernate.App</mainClass>

        </configuration>

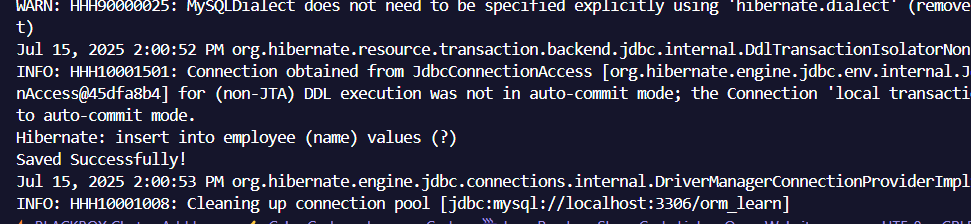
      </plugin>

    </plugins>

  </build>

</project>

**Output**



**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);

}

**Advantages**:

* No session/transaction handling needed
* Very minimal code
* Readable and maintainable

**Code**

**Creating Schema and Table**

CREATE DATABASE orm\_learn;

USE orm\_learn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India'), ('US', 'United States');

**application.properties**

spring.datasource.url=jdbc:mysql://localhost:3306/orm\_learn

spring.datasource.username=root

spring.datasource.password=root

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

spring.jpa.hibernate.ddl-auto=none

spring.jpa.show-sql=true

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.\*;

@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();

}

}

**OrmApplication.java**

package com.cognizant.ormlearn;

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.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.ApplicationContext;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication

@EntityScan("com.cognizant.ormlearn.model")

@EnableJpaRepositories("com.cognizant.ormlearn.repository")

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);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

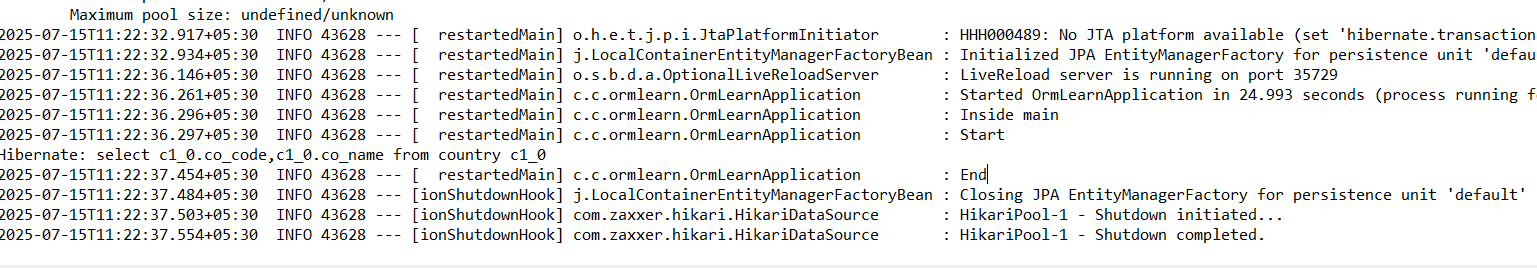
List<Country> countries = countryService.getAllCountries();

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

LOGGER.info("End");

}

}

**Output**