**Spring Data JPA – Quick Example**

Pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.0</version> <!-- or latest stable version -->

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>jpa-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>jpa-demo</name>

<description>Demo project for Spring Data JPA</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Optional: Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<!-- For testing -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

src/main/java/com/example/jpa/model/Product.java

package com.example.jpa.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private double price;

// Constructors, getters, setters

public Product() {}

public Product(String name, double price) {

this.name = name;

this.price = price;

}

// Getters and setters

public Long getId() { return id; }

public String getName() { return name; }

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

public double getPrice() { return price; }

public void setPrice(double price) { this.price = price; }

@Override

public String toString() {

return "Product{" + "id=" + id + ", name='" + name + '\'' + ", price=" + price + '}';

}

}

src/main/java/com/example/jpa/repository/ProductRepository.java

package com.example.jpa.service;

import com.example.jpa.model.Product;

import com.example.jpa.repository.ProductRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public List<Product> getAllProducts() {

return productRepository.findAll();

}

public Product saveProduct(Product product) {

return productRepository.save(product);

}

public void deleteProduct(Long id) {

productRepository.deleteById(id);

}

}

src/main/java/com/example/jpa/JpaDemoApplication.java

package com.example.jpa;

import com.example.jpa.model.Product;

import com.example.jpa.repository.ProductRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class JpaDemoApplication {

public static void main(String[] args) {

SpringApplication.run(JpaDemoApplication.class, args);

}

@Bean

public CommandLineRunner demo(ProductRepository repository) {

return (args) -> {

// Save a few products

repository.save(new Product("Laptop", 999.99));

repository.save(new Product("Smartphone", 699.99));

repository.save(new Product("Headphones", 149.99));

// Fetch all products

System.out.println("Products found with findAll():");

System.out.println("-------------------------------");

for (Product product : repository.findAll()) {

System.out.println(product);

}

System.out.println();

// Fetch an individual product

Product product = repository.findById(1L).orElse(null);

System.out.println("Product found with findById(1L):");

System.out.println("--------------------------------");

System.out.println(product);

System.out.println();

// Update a product

if (product != null) {

product.setPrice(1099.99);

repository.save(product);

System.out.println("Updated product price:");

System.out.println(repository.findById(1L));

System.out.println();

}

// Delete a product

repository.deleteById(2L);

System.out.println("Products after deletion:");

System.out.println("-----------------------");

for (Product p : repository.findAll()) {

System.out.println(p);

}

};

}

}

src/main/resources/application.properties

# H2 Database

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# Hibernate

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.format\_sql=true

# H2 Console (access at http://localhost:8080/h2-console)

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

OUTPUT:

:: Spring Boot :: (v3.2.0)

2025-07-07T00:31:23.538+05:30 INFO 1016 --- [ main] com.example.jpa.JpaDemoApplication : Starting JpaDemoApplication using Java 17.0.7 with PID 1016 (C:\Users\HP\IdeaProjects\jpa\_demo\target\classes started by HP in C:\Users\HP\IdeaProjects\jpa\_demo)

2025-07-07T00:31:23.543+05:30 INFO 1016 --- [ main] com.example.jpa.JpaDemoApplication : No active profile set, falling back to 1 default profile: "default"

2025-07-07T00:31:24.296+05:30 INFO 1016 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.

2025-07-07T00:31:24.381+05:30 INFO 1016 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 74 ms. Found 1 JPA repository interface.

2025-07-07T00:31:25.045+05:30 INFO 1016 --- [ main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]

2025-07-07T00:31:25.129+05:30 INFO 1016 --- [ main] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.3.1.Final

2025-07-07T00:31:25.182+05:30 INFO 1016 --- [ main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled

2025-07-07T00:31:25.743+05:30 INFO 1016 --- [ main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer

2025-07-07T00:31:25.786+05:30 INFO 1016 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...

2025-07-07T00:31:26.049+05:30 INFO 1016 --- [ main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:testdb user=SA

2025-07-07T00:31:26.065+05:30 INFO 1016 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.

2025-07-07T00:31:27.581+05:30 INFO 1016 --- [ main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)

Hibernate:

create table product (

id bigint generated by default as identity,

name varchar(255),

price float(53) not null,

primary key (id)

)

2025-07-07T00:31:27.673+05:30 INFO 1016 --- [ main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'

2025-07-07T00:31:28.605+05:30 INFO 1016 --- [ main] com.example.jpa.JpaDemoApplication : Started JpaDemoApplication in 5.608 seconds (process running for 6.927)

Hibernate:

insert

into

product

(name, price, id)

values

(?, ?, default)

Hibernate:

insert

into

product

(name, price, id)

values

(?, ?, default)

Hibernate:

insert

into

product

(name, price, id)

values

(?, ?, default)

Products found with findAll():

-------------------------------

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

Product{id=1, name='Laptop', price=999.99}

Product{id=2, name='Smartphone', price=699.99}

Product{id=3, name='Headphones', price=149.99}

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

where

p1\_0.id=?

Product found with findById(1L):

--------------------------------

Product{id=1, name='Laptop', price=999.99}

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

where

p1\_0.id=?

Hibernate:

update

product

set

name=?,

price=?

where

id=?

Updated product price:

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

where

p1\_0.id=?

Optional[Product{id=1, name='Laptop', price=1099.99}]

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

where

p1\_0.id=?

Hibernate:

delete

from

product

where

id=?

Products after deletion:

-----------------------

Hibernate:

select

p1\_0.id,

p1\_0.name,

p1\_0.price

from

product p1\_0

Product{id=1, name='Laptop', price=1099.99}

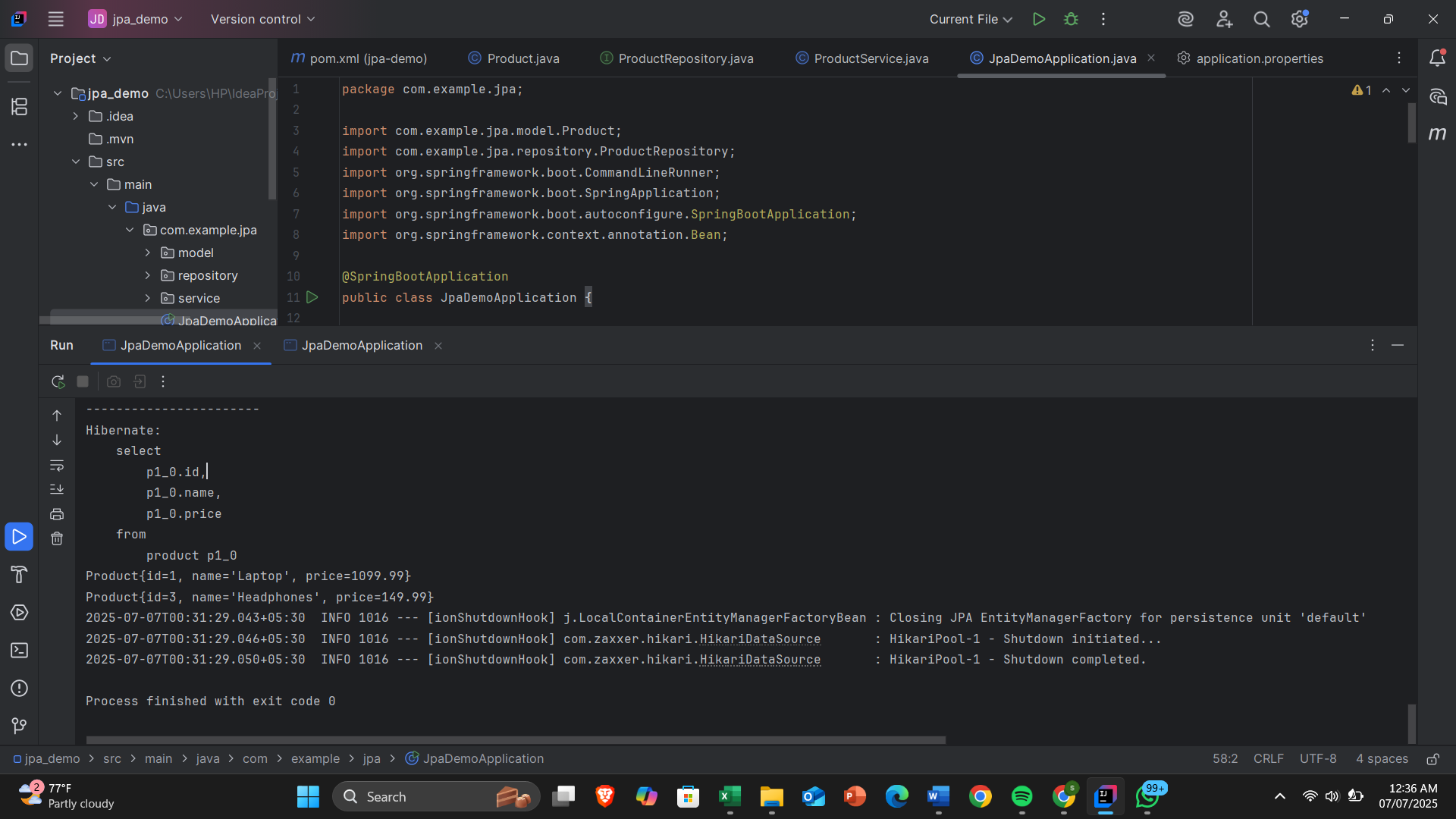
Product{id=3, name='Headphones', price=149.99}

2025-07-07T00:31:29.043+05:30 INFO 1016 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'

2025-07-07T00:31:29.046+05:30 INFO 1016 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...

2025-07-07T00:31:29.050+05:30 INFO 1016 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

Process finished with exit code 0



**JPA vs Hibernate vs Spring Data JPA**

Object-Relational Mapping (ORM) frameworks bridge the gap between object-oriented programming and relational databases. In Java, this is primarily achieved through:

* **JPA (Java Persistence API)** - The standard specification
* **Hibernate** - The most popular implementation of JPA
* **Spring Data JPA** - A higher-level abstraction that simplifies JPA usage

**JPA (Java Persistence API)Definition**

JPA is a Java specification (JSR 338) that standardizes how Java objects are mapped to relational database tables. It provides:

* A set of interfaces (EntityManager, EntityManagerFactory)
* Standard annotations (@Entity, @Table, @Id)
* Query language (JPQL)

**Key Characteristics**

**Specification Only**: Defines standards but doesn't provide implementation

* **Provider Agnostic**: Works with multiple implementations (Hibernate, EclipseLink, etc.)
* **Portable**: Code can work with different JPA providers

**Core Components**

1. **Entity**: A Java class mapped to a database table (@Entity)
2. **EntityManager**: Interface for CRUD operations
3. **Persistence Context**: Managed set of entity instances
4. **JPQL**: Object-oriented query language

**Hibernate**

**Definition**

Hibernate is:

* The most popular JPA implementation
* An ORM framework that predates JPA
* Capable of working with or without JPA annotations

**Key Features**

* Implements all JPA specifications
* Provides additional features beyond JPA:
  + HQL (Hibernate Query Language)
  + Advanced caching mechanisms
  + Custom dialects for different databases
  + Lazy loading and batch processing
* Offers both JPA (EntityManager) and native (Session) APIs

**Architecture**

Hibernate operates through:

1. **SessionFactory**: Creates Session instances
2. **Session**: Main runtime interface
3. **Transaction**: Defines unit of work
4. **ConnectionProvider**: Manages database connections

**Spring Data JPA**

**Definition**

Spring Data JPA is:

* An abstraction layer on top of JPA providers
* Part of the larger Spring Data project
* Designed to reduce boilerplate code

**Key Features**

1. **Repository Abstraction**:
   * JpaRepository interface provides CRUD operations
   * Custom queries can be derived from method names
2. **Reduced Boilerplate**:
   * Eliminates need for DAO implementations
   * Automatic transaction management
3. **Integration**:
   * Seamless with Spring ecosystem
   * Works with multiple JPA providers (defaults to Hibernate)

**Working Mechanism**

* Uses JPA annotations internally
* Generates implementations automatically at runtime
* Provides:
  + @Query for custom queries
  + Pagination and sorting support
  + Auditing capabilities