**1.Spring Data JPA – Quick Example**

**Introduction**

Spring Data JPA is a part of the larger Spring Data framework that simplifies the implementation of Java Persistence API (JPA)–based data access layers. It allows developers to reduce boilerplate code by offering ready-made repository implementations for common database operations such as insert, update, delete, and select. Spring Data JPA integrates seamlessly with Spring Boot and uses Hibernate as the default JPA provider.

This section provides a quick walkthrough of a simple Spring Boot application that demonstrates the use of Spring Data JPA for accessing and managing data.

**Technologies Used**

* Spring Boot
* Spring Data JPA
* Hibernate (JPA Implementation)
* MySQL / H2 Database
* Maven
* IntelliJ IDEA (for development)

**Project Setup**

Spring Data JPA simplifies database interactions in Java applications by reducing boilerplate code through automatic repository implementations. In this example, we will:

* Create a **Student** entity class.
* Use Spring Data JPA to store and retrieve data from an in-memory H2 database.
* Automatically insert sample data **(Alice, Bob)** when the application starts.

**Implementation**

### ****Step 1: Create a Spring Boot Project****

* Create a new Maven project named **student**

**Step 2: Add Dependencies to pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<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>

</plugin>

</plugins>

</build>

### ****Step 3: Configure**** application.properties

Create src/main/resources/application.properties with:

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

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

spring.datasource.username=chandru

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

### ****Step 4: Create**** Student ****Entity Class****

package com.example.studentdemo;

import jakarta.persistence.\*;

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private int marks;

public Student() {}

public Student(String name, String email, int marks) {

this.name = name;

this.email = email;

this.marks = marks;

}

// Getters and Setters

public Long getId() {

return id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public int getMarks() {

return marks;

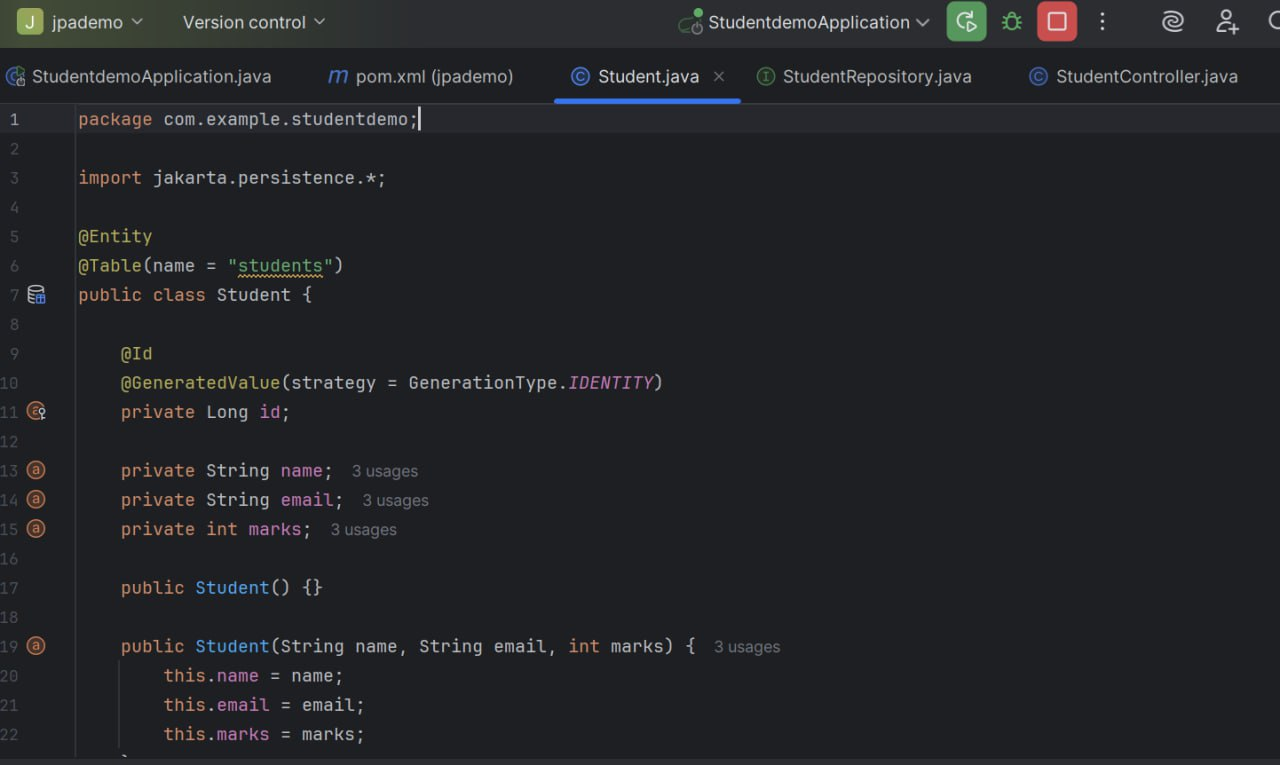
}

public void setMarks(int marks) {

this.marks = marks;

}

}

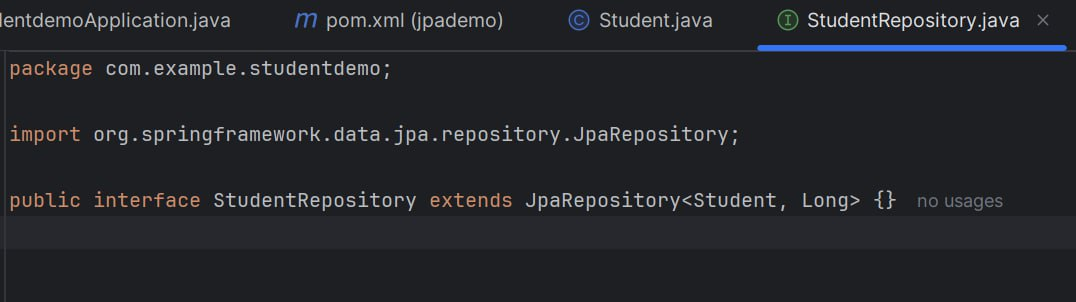


### ****Step 5: Create**** StudentRepository ****Interface****

package com.example.studentdemo;

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

public interface StudentRepository extends JpaRepository<Student, Long> {}



**Step 6: Create REST** StudentController **for student Data access**

package com.example.studentdemo;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/students")

public class StudentController {

@Autowired

private StudentRepository repo;

// POST: Create new student

@PostMapping

public Student createStudent(@RequestBody Student student) {

return repo.save(student);

}

// GET: Retrieve all students

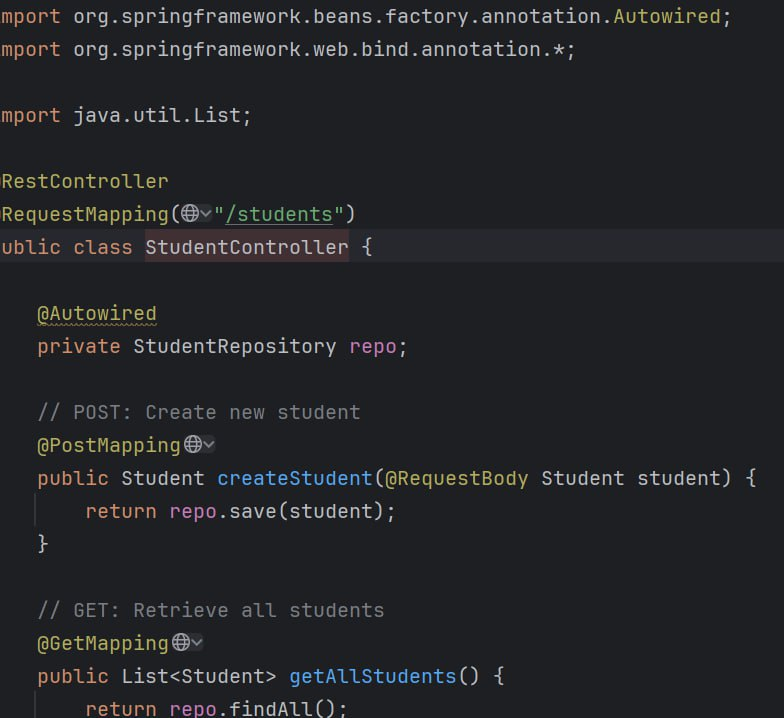
@GetMapping

public List<Student> getAllStudents() {

return repo.findAll();

}

}



**Step 7: Create** StudentdemoApplication.java **to Insert Data**

package com.example.studentdemo;

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 StudentdemoApplication {

public static void main(String[] args) {

SpringApplication.run(StudentdemoApplication.class, args);

}

@Bean

public CommandLineRunner run(StudentRepository repo) {

return args -> {

repo.save(new Student("Alice", "alice@gmail.com", 88));

repo.save(new Student("Bob", "bob@gmail.com", 72));

repo.save(new Student("Charlie", "charlie@gmail.com", 95));

System.out.println("Students saved to the database.");

};

}

}

### ****Step 8: Run and Verify****

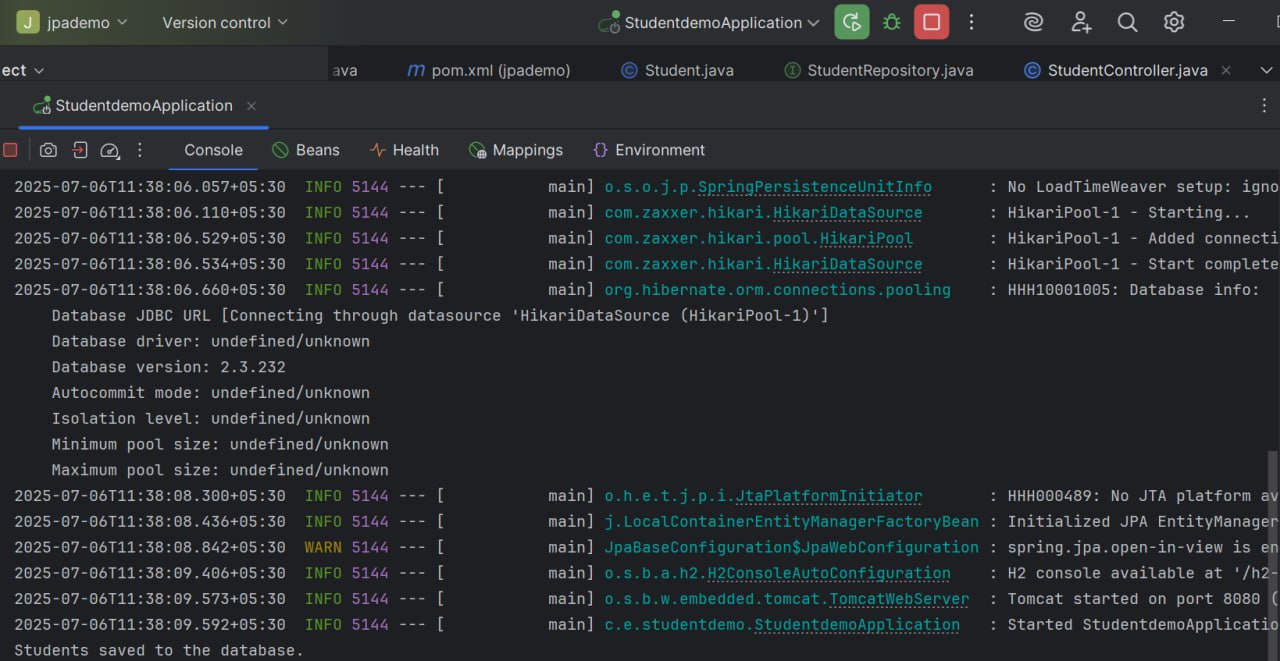
1. **Run the application** – It should start on **http://localhost:8080.**
2. **Open H2 Console** – Go to **http://localhost:8080/h2-console**

* JDBC URL: **jdbc:h2:mem:testdb**
* Username: **chandru**
* No password

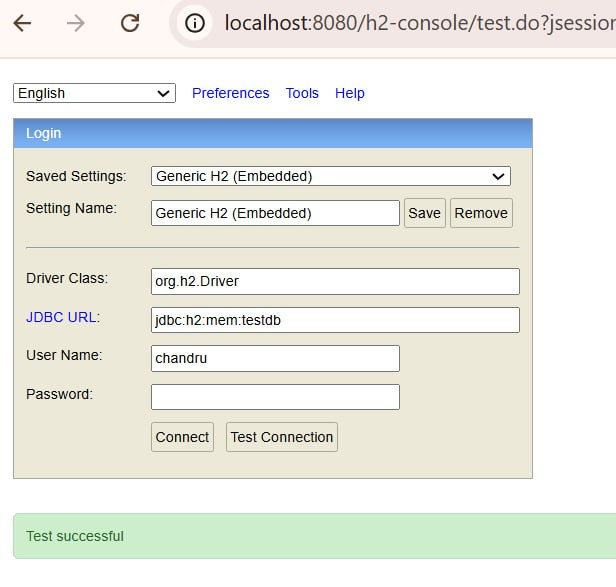
1. **Run SQL Query**:

SELECT \* FROM students;

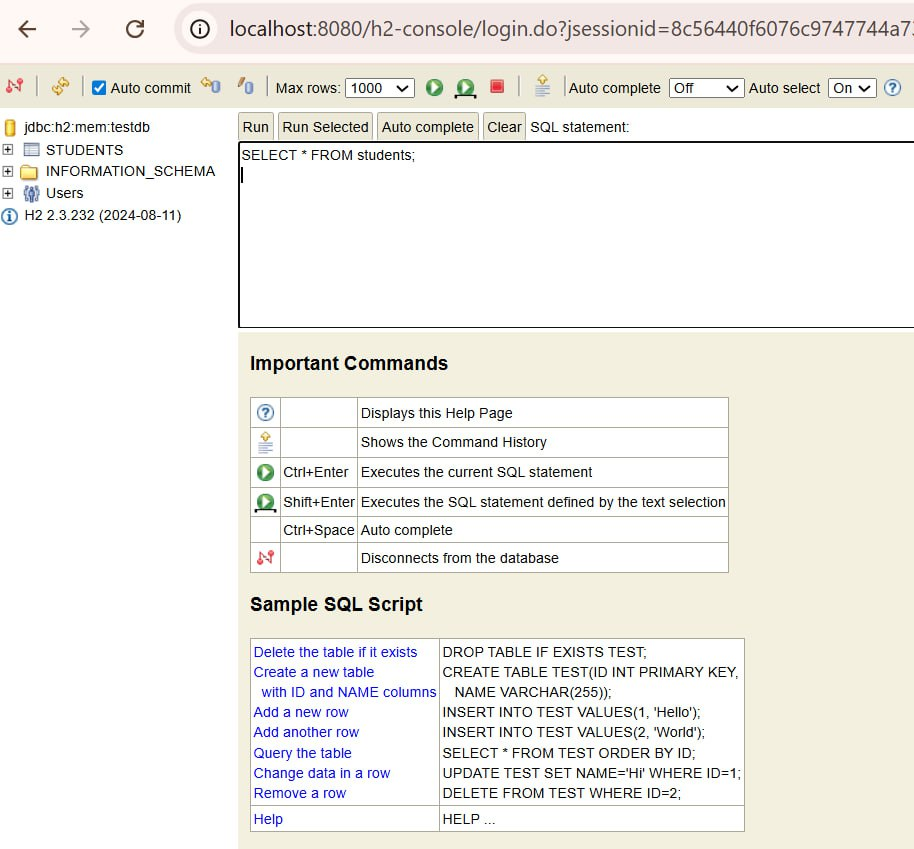
**Run the StudentdemoApplication.java**



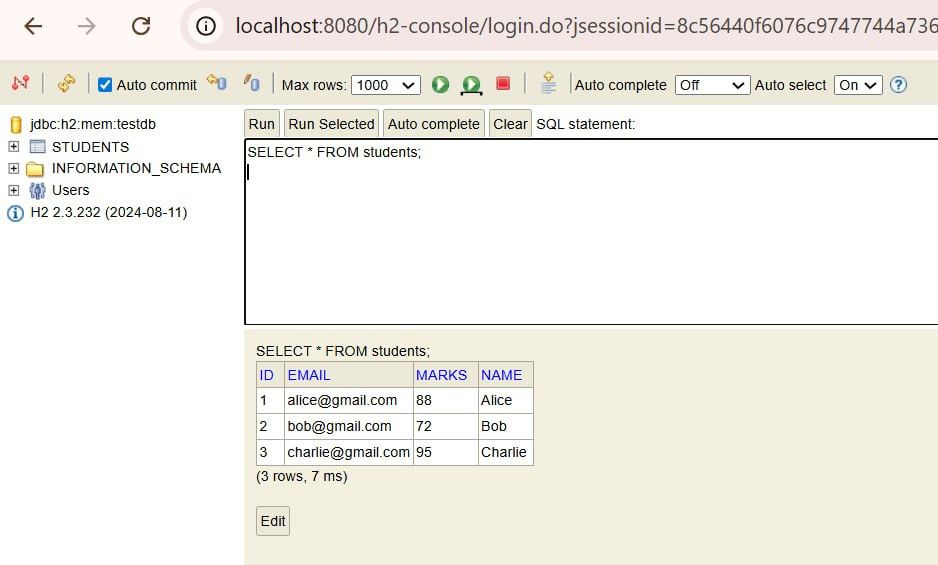
**Open the localhost:8080 & Test Connection**



**Click on Connect & Type the Query**



**Run the Query**



## ****Conclusion****

In this example, we demonstrated how to:

* Set up a Spring Boot application with Spring Data JPA.
* Define a JPA entity (Student) and a repository interface (StudentRepository).
* Insert records into an in-memory H2 database using CommandLineRunner.

This forms the **foundation for building robust database-driven applications** with minimal boilerplate using Spring Data JPA.

**2.Difference between JPA, Hibernate, and Spring Data JPA**

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

1. JPA is a **specification**, not a framework – it defines interfaces and annotations for ORM in Java.
2. It allows Java objects to be mapped to relational database tables using annotations like @Entity, @Id, @Column.
3. Requires an implementation such as **Hibernate** or **EclipseLink** to work.
4. Encourages separation of concerns by abstracting persistence logic from implementation.
5. Provides standard APIs like EntityManager for performing database operations.

**Example (JPA annotations in an entity):**

import jakarta.persistence.\*;

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

**2. Hibernate**

1. Hibernate is a **framework** that implements the JPA specification.
2. Offers additional features like **HQL (Hibernate Query Language)**, caching, and lazy loading.
3. Converts Java classes to database tables and vice versa.
4. Supports automatic schema generation and complex associations (OneToMany, ManyToMany, etc.).
5. Works with both JPA annotations and its own legacy XML or annotations.

**Example (using Hibernate Session API):**

Session session = sessionFactory.openSession();

session.beginTransaction();

Student s = new Student();

s.setName("chandru");

session.save(s);

session.getTransaction().commit();

session.close();

**3. Spring Data JPA**

1. Built on top of JPA; **simplifies CRUD operations** by providing built-in repository interfaces.
2. Eliminates boilerplate code for DAOs (Data Access Objects).
3. Integrates seamlessly with Spring Boot and uses Hibernate as the default provider.
4. Uses interfaces like JpaRepository for custom and derived queries.
5. Supports pagination, sorting, and custom queries via @Query.

**Example (Spring Data JPA repository):**

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification | Framework (JPA implementation) | Framework (built on JPA and Hibernate) |
| **Purpose** | Standard API for ORM | ORM implementation with advanced features | Simplify JPA usage in Spring applications |
| **Boilerplate Code** | Needs EntityManager, manual CRUD | Needs SessionFactory or JPA setup | Minimal code via JpaRepository interface |
| **Provider Required** | Yes (e.g., Hibernate, EclipseLink) | No (it is the provider) | Uses Hibernate (default) as provider |
| **Query Language** | JPQL | HQL + JPQL | Derived queries, JPQL, and native queries |
| **Example Annotation** | @Entity, @Id | Same as JPA, plus Session usage | Uses repository: findByName(String name) |
| **Integration** | Can be used in any Java EE or SE app | Works in Java SE/EE + Spring | Strongly integrated with Spring Boot |
| **Abstraction Level** | High (standard interface only) | Mid (implementation of JPA + custom features) | Higher (abstracts both JPA and Hibernate details) |