Spring Data JPA with Spring Boot & Hibernate Exercises

# Spring Data JPA is a part of the Spring Data family, designed to simplify the implementation of JPA-based repositories. It integrates seamlessly with Spring Boot and Hibernate, providing a powerful abstraction over the JPA API and removing the boilerplate code for data access layers. By using Spring Data JPA, developers can focus more on the business logic rather than dealing with the intricacies of JDBC or JPA entity managers. This exercise demonstrates how to quickly set up Spring Data JPA with Spring Boot, Hibernate, and an in-memory H2 database to perform basic CRUD operations.

**Introduction**

Spring Data JPA simplifies database interactions by abstracting boilerplate code using repository interfaces. When integrated with Spring Boot and Hibernate, it allows fast development with minimal configuration.

# Objective

To quickly build a Spring Boot application using Spring Data JPA and Hibernate that performs CRUD operations on a database.

# Steps

**1. Add Dependencies in pom.xml**

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
</dependency>

**2. Create Entity Class**

@Entity  
public class Student {  
 @Id  
 private int id;  
 private String name;  
  
 // Getters and Setters  
}

**3. Create Repository Interface**

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

**4. Create Main Application**

@SpringBootApplication  
public class JpaExampleApplication implements CommandLineRunner {  
   
 @Autowired  
 private StudentRepository repo;  
  
 public static void main(String[] args) {  
 SpringApplication.run(JpaExampleApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) {  
 repo.save(new Student(1, "Ravi"));  
 repo.findAll().forEach(System.out::println);  
 }  
}

**5. Application.properties**

spring.datasource.url=jdbc:h2:mem:testdb  
spring.jpa.show-sql=true  
spring.h2.console.enabled=true

# Explanation

- Entity defines the table structure.

- Repository provides CRUD methods automatically.

- Spring Boot initializes everything, and the database is in-memory.

# Benefits

- No SQL required for basic operations.

- Faster development with cleaner code.

- Built-in integration with Hibernate.

# Exercise 2: JPA vs Hibernate vs Spring Data JPA

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | JPA (Java Persistence API) | Hibernate | Spring Data JPA |
| **Definition** | Java standard specification for ORM | A JPA implementation + additional features | Abstraction layer over JPA |
| **Origin** | Provided by Oracle (as specification) | Developed by RedHat | Developed by Spring Team |
| **Requires Implementation** | Yes (e.g., Hibernate, EclipseLink) | No, it's a provider | Uses JPA provider under the hood |
| **Boilerplate Code** | Requires EntityManager & queries | Requires session handling | Eliminates most boilerplate via repositories |
| **Complexity** | Higher | Medium | Low |
| **Spring Integration** | Needs manual config | Easily integrated | Out-of-the-box integration |
| **Ease of Use** | Requires more code | Easier than plain JPA | Easiest and quickest for CRUD |

# Conclusion

In this document, we have covered how to quickly set up and use Spring Data JPA with Spring Boot and Hibernate. We saw how simple it is to define entities, create repository interfaces, and execute CRUD operations with minimal setup.

Additionally, we compared JPA, Hibernate, and Spring Data JPA to better understand their individual roles and how they complement each other in building robust and scalable Java applications.

Spring Data JPA provides a clean and productive way to handle persistence in Java applications and is ideal for both beginners and experienced developers aiming for rapid development.