Spring Data JPA with Spring Boot & MySQL

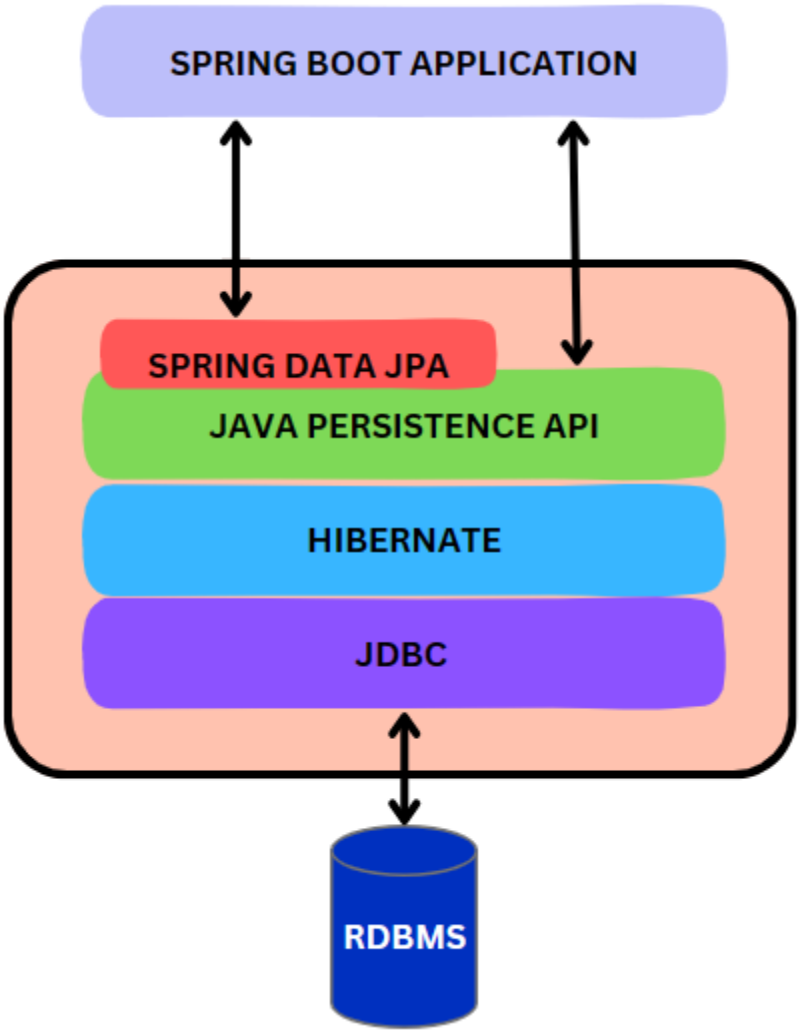
---

1. Overview of Spring Data JPA

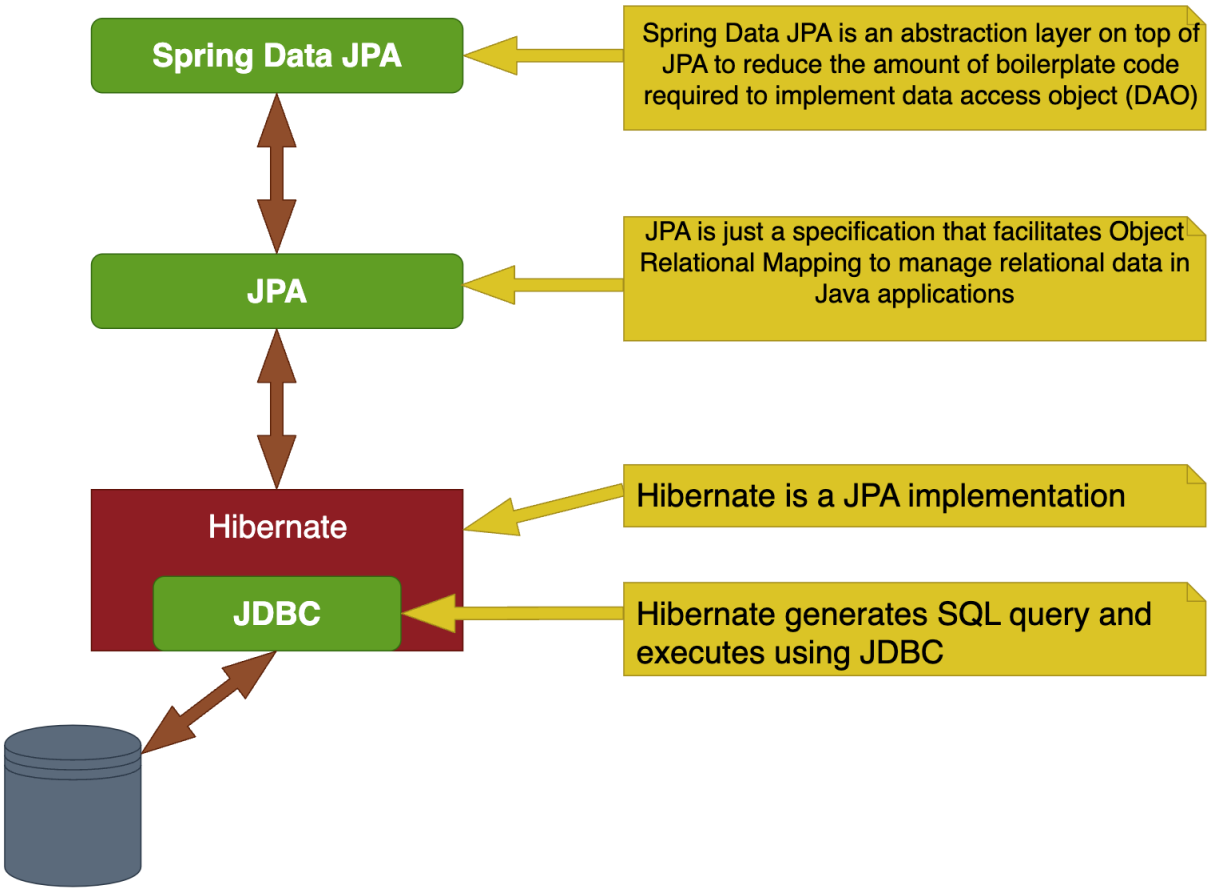
Spring Data JPA is part of the larger Spring Data family, designed to make it easier to implement JPA-based data access layers. It simplifies the development process by eliminating the need for **boilerplate code** that's typically required for data access operations. **Spring Data JPA** uses repositories and abstracts complex queries behind simple method names.

Spring Data JPA simplifies the use of **JPA (Java Persistence API)** by reducing boilerplate and allowing you to perform database operations with minimal code.

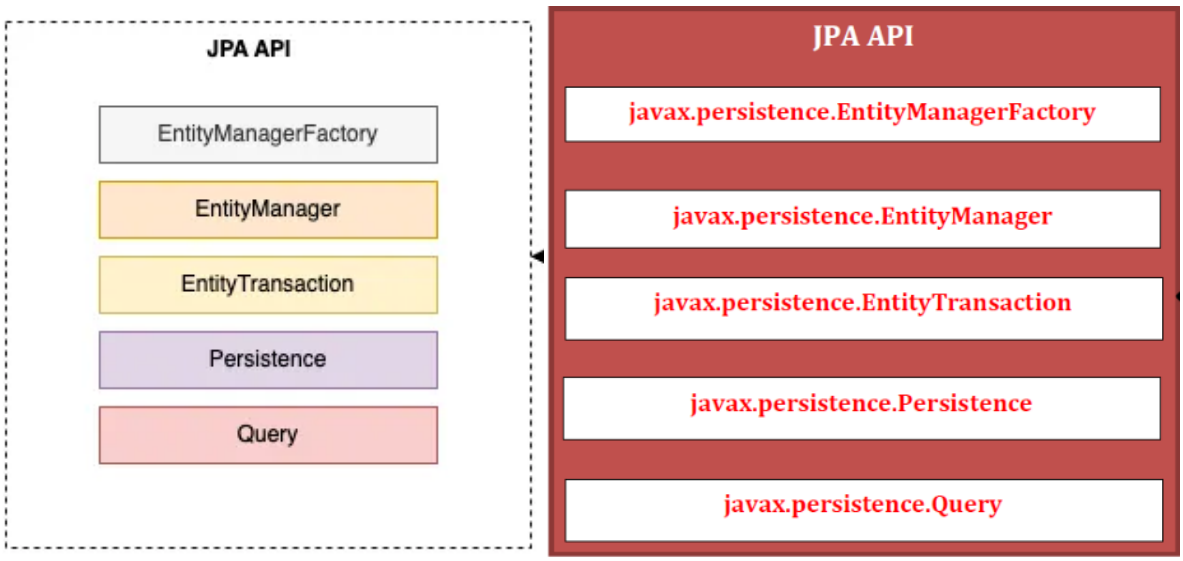
2. Relationship between JPA and Spring Data JPA



Hibernate VS Spring Data JPA



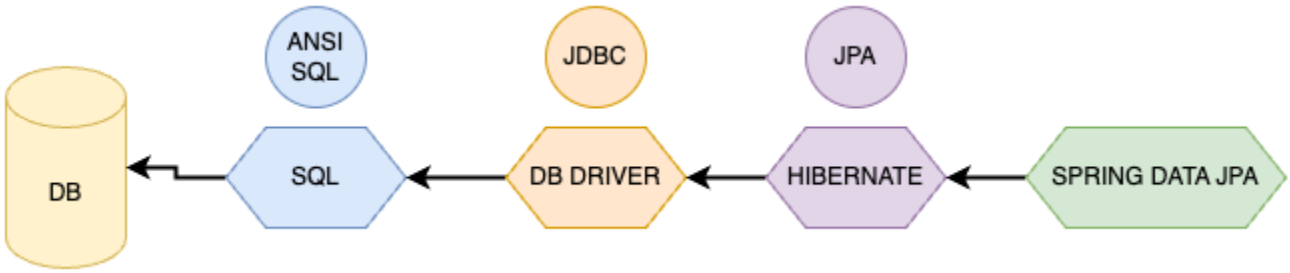
JPA (Java Persistence API) is a specification that standardizes object-relational mapping in Java. It defines how Java objects are mapped to database tables and how **CRUD** operations are performed.



Units	Description
EntityManagerFactory	Factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager	It is an interface; it manages the persistence operations on objects. It works like a factory for query instances.
Entity	Entities are the persistence objects, stored as records in the database.
EntityTransaction	It has a one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by the EntityTransaction class.
Persistence	This class contains static methods to obtain EntityManagerFactory instance.
Query	Each JPA vendor implements this interface to obtain relational objects that meet the criteria.

Spring Data JPA is a library that builds on **top of JPA**, providing additional features like repository interfaces, query derivation from method names, and integration with Spring Boot for easy setup.

How Persistence Operation works



- **JPA is a standard (a specification).**
- **Hibernate is a common implementation of JPA.**
- **Spring Data JPA is a Spring module that wraps JPA to simplify usage.**

3. Advantages of Using Spring Data JPA in Spring Boot

- **Rapid Development:** Reduces time to write repetitive CRUD operations.
- **Less Boilerplate:** Automatically implements repository methods.
- **Integration:** Seamlessly integrates with Spring Boot.
- **Query Methods:** Supports powerful, expressive queries with method naming conventions.
- **Pagination & Sorting:** Built-in support for paging and sorting of query results.

---

4. Creating a Spring Boot Project

To get started, you can use Spring Initializr to generate a Spring Boot project. This tool allows you to select dependencies and project metadata easily. After generation, you'll get a structured project with the necessary configurations to begin development.

Spring Data JPA simplifies database access by using repository interfaces. You don't have to write **SQL** or **implement DAO classes** — just define an interface and **Spring Data JPA** handles the rest.

Example:

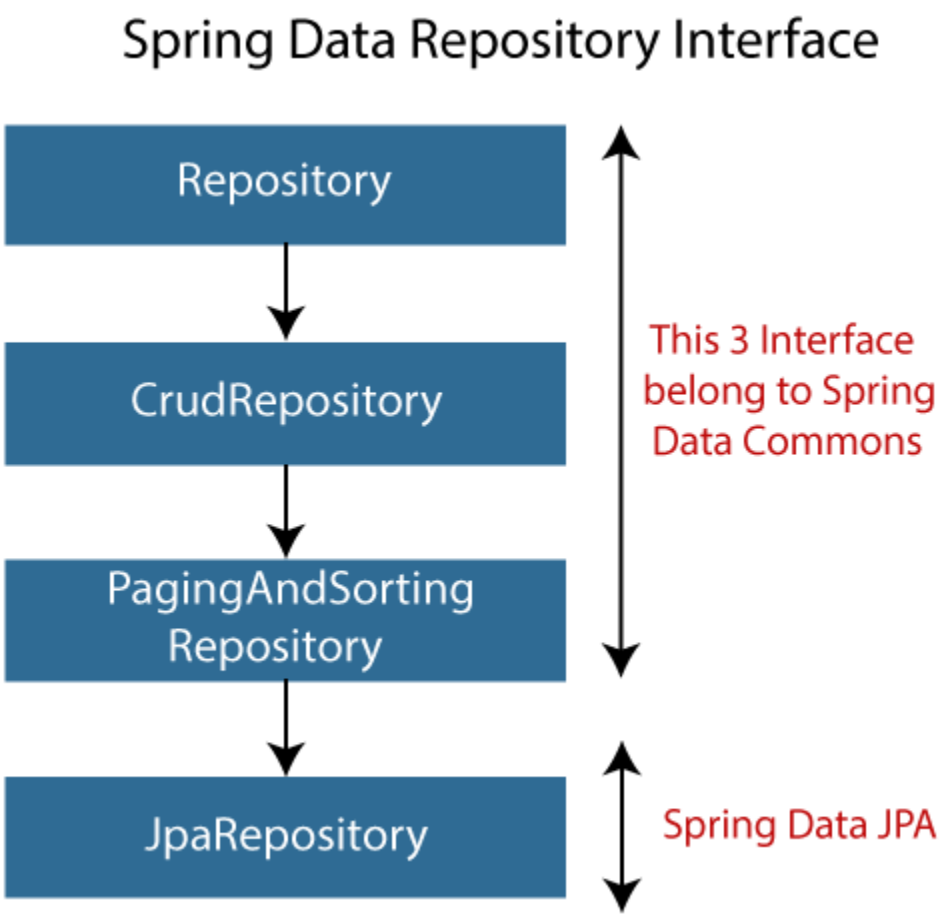
```
import org.springframework.data.jpa.repository.JpaRepository;

import com.cts.entity.User;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

This one line gives you basic CRUD operations like save(), findById(), findAll(), and delete().

This gives you full **CRUD s**upport out-of-the-box.



---

2. Relationship: JPA vs Spring Data JPA

- **JPA** is just a specification (**it doesn't do anything on its own**).
- **Hibernate** is an implementation of **JPA.**
- **Spring Data JPA** uses **Hibernate** (**or any JPA provider**) and makes it super easy to integrate with Spring Boot.

Example Entity:

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
```

```java
public class User {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String username;
	private String email;
```

---

3. Why Use Spring Data JPA in Spring Boot?

- **Simplifies repository code** (**no DAO classes needed**)
- **Auto-generates CRUD and query methods**
- **Integrates cleanly with MySQL or any other relational database**
- **Supports paging, sorting, and custom queries**

---

4. Setting Up a Spring Boot Project

1. Go to Spring Initializr/Spring Starter Project
2. Choose:
    - **Project**: Maven
    - **Dependencies**:
        - **Spring Web**
        - **Spring Data JPA**
        - **MySQL Driver**
3. **Generate and open the project in your IDE (STS, IntelliJ, Eclipse, etc.)**

---

5. Add MySQL Dependency

In pom.xml (Maven):

```xml
<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>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
```

---

6. Configure MySQL Connection

In src/main/resources/application.properties:

```
spring.application.name=SpringDataJPA

server.port=2024

# ================================
# = DATABASE CONNECTION
# ================================
spring.datasource.url=jdbc:mysql://localhost:3306/cts
spring.datasource.username=root
spring.datasource.password=root
```

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

# ================================
# = JPA / HIBERNATE
# ================================
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate
.dialect.MySQL8Dialect

# (Optional) Format SQL output
spring.jpa.properties.hibernate.format_sql=true
```

Make sure:

- **MySQL is installed and running**
- **The database exists**
- **The user has access to it**