Data Access with Spring Boot using Spring Data JPA

1. Introduction to Spring Data JPA

- **Spring Data JPA** is a part of the Spring ecosystem used for **data persistence**.
- It simplifies the development of **data access layers** using **JPA (Java Persistence API)**.
- Reduces boilerplate code and enables easy database interactions using **repositories**.

2. ORM Basics

- **Object-Relational Mapping (ORM)** maps **Java objects (entities)** to **database tables**.
- Annotations like `@Entity`, `@Id`, and `@Column` help define this mapping.

Example:

```
@Entity
public class Product {
  @Id
  @GeneratedValue
  private Long id;

  private String name;
  private double price;
}
```

3. Configuring Databases (H2 and MySQL)

For H2 (in-memory database)

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

For MySQL

```
spring.datasource.url=jdbc:mysql://localhost:3306/demo
spring.datasource.username=root
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

4. Creating Entity Classes

- Entity classes represent **database tables**.
- Annotated with `@Entity`, each instance maps to a **row** in the table.

```
@Entity
@Table(name = "employees")
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String name;
  private String department;
}
```

5. Repository Interfaces

- Extend `JpaRepository` or `CrudRepository` to handle CRUD operations.

  ```
  public interface EmployeeRepository extends
  JpaRepository<Employee, Long> {
  }
  ```

- You get built-in methods like:
    - `findAll()`
    - `findById()`
    - `save()`
    - `deleteById()`

6. Custom Query Methods

- Spring Data JPA supports **method name queries**:

```
List<Employee> findByDepartment(String department);
```

- And **custom JPQL queries** using `@Query`:

```
@Query("SELECT e FROM Employee e WHERE e.department = ?1")
List<Employee> getEmployeesByDept(String department);
```

7. Controller-Service-Repository Architecture

- **Controller** handles HTTP requests (`@RestController`).
- **Service** contains business logic (`@Service`).
- **Repository** interacts with DB (`@Repository`).

Structure:

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService service;
    @GetMapping
    public List<Employee> getAll() {
        return service.getAllEmployees();
    }
}


@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository repo;

    public List<Employee> getAllEmployees() {
        return repo.findAll();
    }
}
```

8. JPQL (Java Persistence Query Language)

- Object-oriented query language similar to SQL.
- Operates on **entity objects** instead of tables.

Example:

```
@Query("SELECT e FROM Employee e WHERE e.name LIKE %:name%")
List<Employee> searchByName(@Param("name") String name);
```

9. Advantages of Spring Data JPA

- Rapid development using **minimal code**.
- Integration with Spring Boot is **seamless**.
- Built-in support for **pagination**, **sorting**, and **custom queries**.
- Supports both in-memory and persistent databases.

10. CRUD Example with H2

```
// POST
@PostMapping("/add")
public Employee addEmployee(@RequestBody Employee emp) {
    return employeeRepository.save(emp);
}

// GET
@GetMapping("/{id}")
public Employee getEmployee(@PathVariable Long id) {
    return employeeRepository.findById(id).orElse(null);
}
```

Summary

Concept	**Description**
ORM	Maps Java objects to DB tables
Entity	Annotated class that represents DB table
Repository	Interface to access DB
JPQL	Object-based query language
Spring Data JPA	Module to simplify JPA in Spring
DB Support	Easily switch between H2 and MySQL