04 - JPA and Database Interaction and JPQL

Table of Contents

Agenda:

- What is JDBC and Hibernate
- Intro to JPA (Java Persistence API)
- Hands-on with JPA
- Setting up Database
- Connecting with Database using application.properties file
- JPQL (Java Persistence Query Language)

JDBC (Java Database Connectivity)

Imagine you need to fetch water from a well. Using JDBC is like manually drawing water with a bucket every time you need it. You have to do all the work yourself, but you have complete control over the process.

Key Points:

- Direct Database Access: You write SQL queries directly and handle database connections manually.
- More Code: Requires more boilerplate code to manage connections, execute queries, and process results.
- Fine-Grained Control: Provides direct control over database operations but can be error-prone and repetitive.

What is Hibernate?

Now, imagine you have a smart assistant who knows where the well is and can fetch water for you whenever you need it. This assistant makes the process easier and faster, and you don't have to worry about the details.

Key Points:

- ORM Tool: Maps Java objects to database tables, reducing the need for writing SQL queries.
- Less Code: Simplifies database operations with less boilerplate code.
- Automatic Mapping: Handles the conversion between Java objects and database tables automatically.

JDBC	Hibernate
It is a low-level API for accessing databases in Java.	It is a high-level framework for Java
JDBC requires developers to write a lot of code to handle database transactions	Hibernate automatically generates the necessary SQL code
Error-handling needs to be done manually by the developer	Error-handling is automated by the framework
It is faster than Hibernate because of its direct interaction with the database	Slower than JDBC due to the overhead of ORM framework
Maintenance is difficult	Maintenance is easier

What is JPA?

Think of JPA as a set of guidelines or rules that define how your smart assistant (like Hibernate) should fetch water from the well. JPA provides the blueprint, while Hibernate is one of the assistants that follow these guidelines.

Key Points:

- Specification: JPA is not a tool but a set of standards for ORM in Java.
- Framework Agnostic: Multiple implementations can follow JPA standards, such as Hibernate, EclipseLink, etc.
- Annotations: Uses annotations to define how Java objects map to database tables.

```
import javax.persistence.Entity;
import javax.persistence.Table;
```

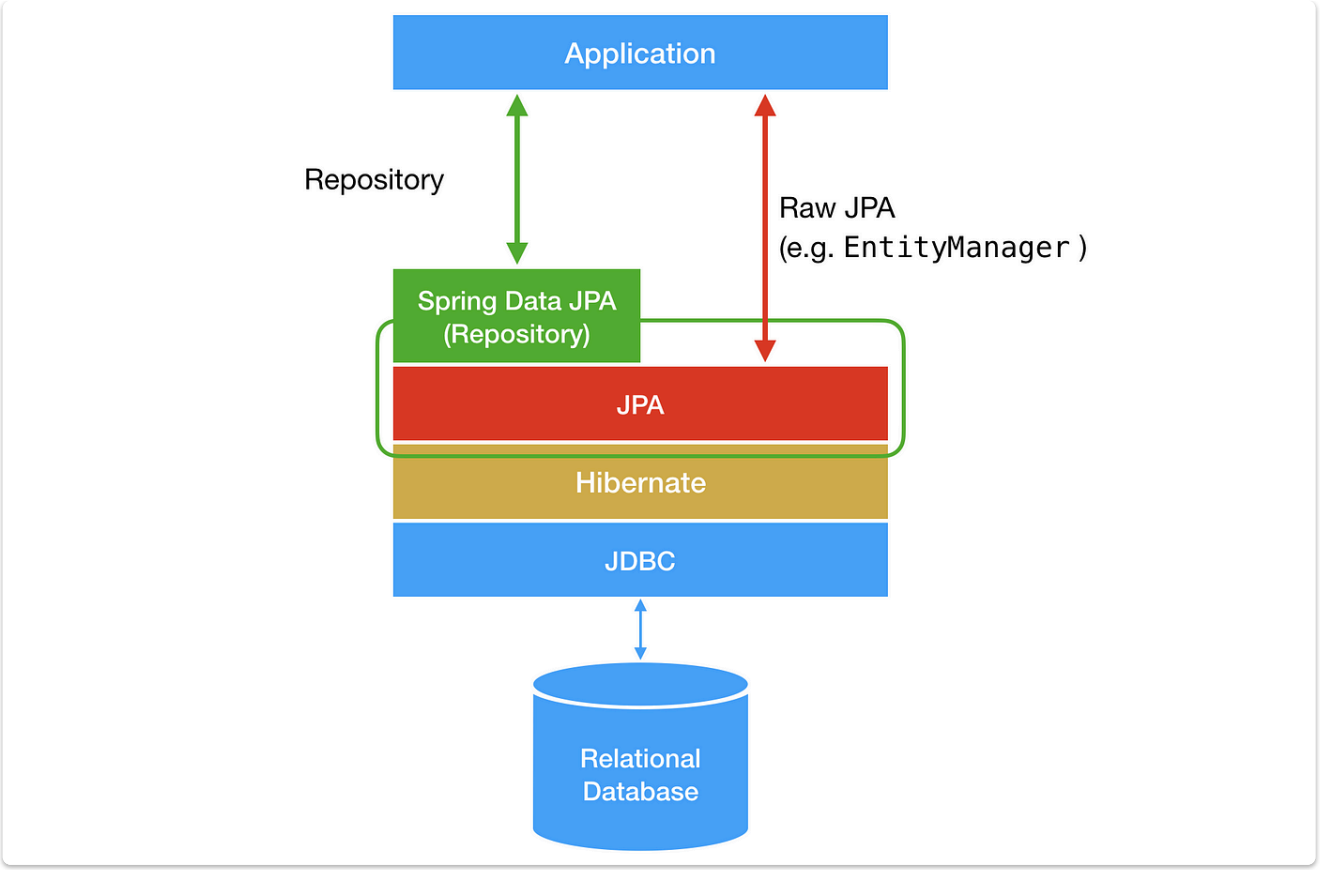
```java
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

@Entity
@Table(name = "books")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    // Getters and setters
}
```

Explanation:

- `@Entity`: Tells JPA that this class is an entity and should be mapped to a database table.
- `@Table(name = "books")`: Specifies the name of the database table to which this entity is mapped.
- `@Id`: Marks the primary key of the entity.
- `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Specifies that the primary key value should be generated automatically.

Summary:

- JDBC: Like manually drawing water from a well. Direct, detailed control over database interactions but requires more effort and code.
- Hibernate: Like having a smart assistant fetch water for you. Simplifies database operations by automatically mapping Java objects to tables.
- JPA: Like the set of guidelines for how the assistant should fetch water. Provides a standard way of defining ORM, which Hibernate and other tools implement.



Connecting with Database using application.properties file

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/your_database_name
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
```

What are Class-Based Views?

Class-based views are a way to define the behavior of views in Spring MVC using Java classes instead of methods. This approach allows for better organization and reuse of code.

Before:

```
@Controller
public class MyController {

    @GetMapping("/greet")
    public String greet() {
        return "Hello, World!";
```

```
    }
}
```

After class based view:

```
@Controller
@RequestMapping("/greet") // this refers to class based view
public class GreetController {

    @GetMapping
    public String greet() {
        return "Hello, World!";
    }
}
```

What is JPQL?

- A platform-independent object-oriented query language defined as part of the JPA specification.
- Used to create queries against entities stored in a relational database.
- It resembles SQL in syntax but operates on Java objects (entities) and their properties, not directly on database tables and columns.

Advantages: WHYYYY>>>???????

- Database Independence: Portable across different database vendors.
- Object-Oriented: Works directly with your entity classes.
- Integration with JPA: Seamlessly integrates with other JPA features like entity managers and repositories.

Writing JPQL Queries

- Basic Structure

```
SELECT [DISTINCT] <select_expression>
FROM <from_clause>
[WHERE <where_clause>]
[GROUP BY <group_by_clause>]
[HAVING <having_clause>]
[ORDER BY <order_by_clause>]
```

```
- Select Expressions: Can be entity objects, single attributes, or
aggregate functions.
- From Clause: Specifies the entity class(es) to query.
- Where Clause: Filters the results based on conditions.

- Group By, Having, Order By: Similar to SQL, used for aggregation,
filtering on aggregated data, and sorting results.
```

Practical Examples and Use Cases

1. **Simple Select:**

```
SELECT c FROM Customer c  -/*Retrieves all `Customer` entities*/
```

2. **Select Specific Fields:**

```
SELECT c.name, c.email FROM Customer c  -/*(Retrieves only the name and
email of all `Customer` entities)*/
```

3. **Filtering with WHERE:**

```
SELECT c FROM Customer c WHERE c.age > 30. -/*(Retrieves `Customer`
entities older than 30)*/
```

4. **Using Parameters:**

```
SELECT c FROM Customer c WHERE c.name = :name.  -/*(Retrieves `Customer`
entity with a specific name, passed as a parameter `:name`)*/
```

5. **Joining Entities:**

```
SELECT o FROM Order o JOIN o.customer c WHERE c.name = 'John Doe' -/*
(Retrieves all orders (`Order` entities) for a customer named 'John
Doe')*/
```

6. **Aggregation:**

```
SELECT AVG(p.price) FROM Product p. -/*(Calculates the average price of
all products)*/
```

Example

```java
public interface UserRepository extends JpaRepository<User,Long> {

    User findByEmail(String email);

    @Query("SELECT u FROM User u WHERE u.name LIKE :character%")
    List<User> findUsersStartingWithA(Character character);
}
```