1. - ORM là gì ? Sử dụng ORM mang lại lợi ích như thế nào cho ứng dụng. Cơ chế hoạt động của ORM như thế nào? So sánh performance của việc sử dụng ORM vs JDBC.

1.1 Định nghĩa : là kỹ thuật lập trình dùng để ánh xạ các bảng trong cơ sở dữ liệu thành các thực thể và các cột là thuộc tính , giúp tương tác với dữ liệu mà không cần viết câu lệnh SQL.

1.2 Lợi ích sử dụng ORM

* **Dễ đọc và dễ bảo trì**: Viết code với ORM thường dễ đọc hơn và sát với ngôn ngữ lập trình, dễ bảo trì hơn so với code SQL thuần.
* **Tự động hoá** : tự động hoá phần CRUD trên data.
* Quản lý qun hệ giữa cáccác bảng : ORM tự dộng quản lý mối quan hệ onetoone, onetomany và manytomany.

1.3 Cơ chế hoạt động

* Các class khai báo với @Entity được ORM xác định và ánh xạ đến bảng tương ứng trong cơ sở dữ liệu.
* Khi thực hiện các thao tác CRUD hoặc thao tác khác ORM sẽ tạo ra những câu kệnh SQL tương ứng và giao tiếp với cơ sở dữ liệu .

1.4 So sánh peformence giữa JDBC và ORM :

- JDBC nhanh hơn do viết câu lệnh sql trực tiếp còn ORM phải thông qua 1 lớp trung gian để dịch câu sql.

1. Spring JPA có phải là 1 triển khai của ORM hay không ?

* Spring Data JPA là một triển khai của ORM (Object-Relational Mapping) dựa trên JPA (Java Persistence API).
* . Spring Data JPA sử dụng Hibernate (một ORM framework phổ biến) làm trình cung cấp JPA mặc định, giúp lập trình viên dễ dàng làm việc với dữ liệu trong cơ sở dữ liệu mà không cần viết quá nhiều truy vấn SQL thủ công.
* Spring Data JPA hỗ trợ các thao tác như truy vấn, lưu trữ, cập nhật và xóa dữ liệu thông qua các repository interface.

2.1 Tìm hiểu các loại quan hệ trong JPA : Many To One, One To Many, Many To Many.

* **Quan hệ Many-to-One**
  + **Đặc điểm**: Trong quan hệ Many-to-One, nhiều thực thể con sẽ liên kết với một thực thể cha duy nhất.
  + **Ví dụ**: Nhiều đơn hàng (Order) có thể thuộc về một khách hàng (Customer).
  + **Triển khai**: Để khai báo quan hệ này trong JPA, sử dụng annotation @ManyToOne trong thực thể con.
* **Quan hệ One-to-Many**
  + **Đặc điểm**: Quan hệ One-to-Many là chiều ngược lại của Many-to-One, một thực thể cha có thể có nhiều thực thể con liên kết.
  + **Ví dụ**: Một khách hàng (Customer) có thể có nhiều đơn hàng (Order).
  + **Triển khai**: Sử dụng annotation @OneToMany trong thực thể cha. Thường đi kèm với mappedBy để chỉ định thực thể con đang sở hữu quan hệ.
* **Quan hệ Many-to-Many**
  + **Đặc điểm**: Trong quan hệ Many-to-Many, một thực thể có thể liên kết với nhiều thực thể khác và ngược lại.
  + **Ví dụ**: Một sinh viên (Student) có thể đăng ký nhiều khóa học (Course), và một khóa học cũng có thể có nhiều sinh viên đăng ký.
  + **Triển khai**: Dùng annotation @ManyToMany ở cả hai bên thực thể, có thể sử dụng @JoinTable để tùy chỉnh bảng trung gian lưu thông tin quan hệ.

2.2 Các loại Cascade type, Fetch type trong JPA

2.2.1 CascadeType :

* Định nghĩa : CascadeType cho phép bạn xác định cách các hành động trên một thực thể chính sẽ ảnh hưởng tới các thực thể liên quan khác.
* Các loại cascadeType :
  + CascadeType.ALL: Áp dụng tất cả các loại hành động (PERSIST, MERGE, REMOVE, REFRESH, DETACH).
  + CascadeType.PERSIST: Khi thực thể chính được lưu (persist), thực thể liên quan cũng sẽ được lưu.
  + CascadeType.MERGE: Khi thực thể chính được hợp nhất (merge), thực thể liên quan cũng sẽ được hợp nhất.
  + CascadeType.REMOVE: Khi thực thể chính bị xóa (remove), thực thể liên quan cũng sẽ bị xóa.
  + CascadeType.REFRESH: Khi thực thể chính được làm mới (refresh), thực thể liên quan cũng sẽ được làm mới.
  + CascadeType.DETACH: Khi thực thể chính bị tách (detach) khỏi EntityManager, thực thể liên quan cũng sẽ bị tách ra.

2.2.2 Fetch Type

* Định nghĩa : Xác định khi nào các thực thể liên quan sẽ gọi từ cơ sở dữ liệu , các 2 loại Fetch :
  + **FetchType.EAGER**: Tải dữ liệu ngay lập tức khi thực thể chính được tải. Điều này có nghĩa là các thực thể liên quan sẽ được nạp đầy đủ ngay lập tức trong một truy vấn duy nhất.
  + **FetchType.LAZY**: Tải dữ liệu chậm, tức là dữ liệu sẽ không được tải cho đến khi nó thực sự được truy cập. Đây là lựa chọn mặc định cho các mối quan hệ OneToMany và ManyToMany.

2.3. Tìm hiểu cách convert DTO -> Entity và ngược lại sử dụng 2 cách : dùng thư viện và sử dụng java reflection.

2.3.1 Dùng thư viện

Covert DTO -> Entity và ngược lại có rất nhiều thư viện hỗ trợ việc này tiêu biểu nhất là Mapstruct đây là lớp dễ sử dụng và có tốc độ convert nhanh.

VD sử dụng :

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface UserMapper {

UserMapper INSTANCE = Mappers.getMapper(UserMapper.class);

// Convert DTO -> Entity

UserEntity toEntity(UserDTO userDTO);

// Convert Entity -> DTO

UserDTO toDto(UserEntity userEntity);}

Trong ví dụ trên ta dùng mapper để convert giữa 2 kiểu DTO -> Entity và ngược lại.

2.3.2 Sử dụng Java Reflection.

Java Reflection cho phép chuyển đổi giữa DTO và Entity mà không cần thêm thư viện, nhưng phức tạp hơn.

Cách triển khai :

Bước 1 : tạo 1 class ánh xa cho cả DTO và Entity .

Bước 2 : sử dụng khi cần convert

1. Advance

3.1 Native query

* Định nghĩa : một dạng truy vấn SQL thuần, cho phép bạn viết các câu lệnh SQL trực tiếp thay vì sử dụng JPQL, được dùng khi truy vấn sql phức tạp và tăng hiệu suất .
* Các cách sử dụng native query
  + Sử dụng @Query với navtiveQuery = true
    - Vd : @Query(value = "SELECT \* FROM users WHERE email = :email", nativeQuery = true);
* Sử dụng EntityManager khi bạn muốn kiểm soát nhiều hơn hoặc thực hiện các truy vấn phức tạp.
  + Vd : Query query = entityManager.createNativeQuery(sql, User.class);
* Sử dụng khi :
  + Truy vấn dữ liệu lớn có độ phức tạp cao .
  + Muốn cải thiện hiệu suất truy vấn .

3.2 Specification, Paging.

3.2.1 Specification

* Specification trong Spring là một tiêu chuẩn (criterion) để tạo các truy vấn động trong cơ sở dữ liệu mà không cần viết SQL thủ công.
* Dùng để xác định các diều kiện cho câu truy vấn động sử dụng jpa
* Cách triển khai
  + tạo các hàm chứa điều kiện truy vấn
  + Dùng specification.where() để thêm điều kiện
  + gọi và sử dụng trong truy vấn Vd findAll(spec) hàm này sẽ query lấy tất cả bản ghi mà điều kiện của chúng đúng với spec truyền vào .

3.2.2 Paging

* Paging trong Spring giúp giới hạn số lượng bản ghi trả về, từ đó giảm tải cho hệ thống khi xử lý dữ liệu lớn.
* Cách triển khai :
  + Dùng Pageable để xác tang hiện tại và số luọng bản ghi trong trang.
  + Dùng Page để lấy dữ liệu .
  + Thêm pageable vào page để trả về số lượng bản ghi theo yêu cầu .
* Mục đích sử dụng :
  + Giảm tải hệ thống khi thực hiện query vào những bảng lớn .

3.3 Lấy dữ liệu khi JOIN nhiều bảng sử dụng JPA (mỗi bảng lấy 1 tới 2 trường thông tin)

Để thực hiện việc join bảng và lấy thông tin ta dùng JPQL hoặc @query .