ORM

**1. ORM là gì? Lợi ích và cơ chế hoạt động của ORM**

* **ORM (Object-Relational Mapping)** là kỹ thuật ánh xạ giữa các đối tượng trong mã nguồn và các bảng trong cơ sở dữ liệu quan hệ, giúp lập trình viên làm việc với các đối tượng thay vì SQL trực tiếp. Các thư viện ORM (như Hibernate) tự động chuyển đổi các thao tác trên đối tượng thành câu lệnh SQL và thực thi trên cơ sở dữ liệu.

**Lợi ích của việc sử dụng ORM:**

* **Tăng năng suất phát triển**: Tạo điều kiện cho lập trình viên làm việc với các đối tượng mà không cần viết SQL phức tạp.
* **Dễ bảo trì**: Giảm sự phụ thuộc vào chi tiết của từng loại cơ sở dữ liệu.
* **Tính linh hoạt**: Các thao tác CRUD và truy vấn dễ dàng thực hiện với các API của ORM.

**Cơ chế hoạt động của ORM:**

* ORM quản lý các đối tượng từ ứng dụng và ánh xạ chúng tới các bảng cơ sở dữ liệu.
* **Phiên dịch các thao tác** trên đối tượng thành câu lệnh SQL và ngược lại.
* **Quản lý cache** và xử lý các mối quan hệ giữa các đối tượng để tăng hiệu năng và sự nhất quán dữ liệu.

**So sánh hiệu năng ORM vs JDBC:**

* **ORM**: Có thể chậm hơn JDBC một chút trong các truy vấn phức tạp do phải ánh xạ và quản lý các đối tượng. Tuy nhiên, ORM có thể được tối ưu hóa bằng cache và fetch type.
* **JDBC**: Truy vấn SQL trực tiếp nên có hiệu năng cao hơn, nhưng yêu cầu nhiều mã nguồn thủ công và dễ dẫn tới lỗi bảo mật (nếu không xử lý tốt SQL Injection).

**2. Spring JPA có phải là một triển khai của ORM không?**

* **Spring Data JPA** không phải là một triển khai của ORM mà là một **framework giúp tích hợp** JPA (Java Persistence API) trong Spring. Spring Data JPA hỗ trợ các thao tác CRUD, truy vấn tùy chỉnh dễ dàng và quản lý dữ liệu hiệu quả.
* Spring Data JPA sử dụng Hibernate như một triển khai ORM mặc định.

**3. Các loại quan hệ trong JPA**

* **One-To-Many**: Một thực thể có thể có nhiều thực thể liên quan (VD: Một User có nhiều Orders).

@OneToMany(mappedBy = "user")

private List<Order> orders;

* **Many-To-One**: Nhiều thực thể có thể tham chiếu tới một thực thể (VD: Nhiều Orders thuộc về một User).

@ManyToOne

@JoinColumn(name = "user\_id")

private User user;

* **Many-To-Many**: Nhiều thực thể có thể có mối quan hệ với nhiều thực thể khác (VD: Một User có nhiều Roles và một Role có nhiều Users).

@ManyToMany

@JoinTable(name = "user\_roles",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "role\_id"))

private Set<Role> roles;

**4. Cascade Type và Fetch Type trong JPA**

* **Cascade Type**: Xác định hành vi cascade giữa các thực thể.
  + **ALL**: Thực hiện tất cả các thao tác cascade (save, update, delete).
  + **PERSIST**: Chỉ cascade thao tác lưu.
  + **MERGE**: Cascade thao tác merge khi cập nhật.
  + **REMOVE**: Cascade thao tác xóa.
  + **REFRESH**: Tải lại thực thể từ database.
  + **DETACH**: Tách thực thể ra khỏi context hiện tại.
* **Fetch Type**:
  + **EAGER**: Tải toàn bộ dữ liệu ngay lập tức.
  + **LAZY**: Chỉ tải dữ liệu khi cần (thường dùng với các collection lớn để tối ưu).

**5. Cách chuyển đổi giữa DTO và Entity**

* **Dùng thư viện**:
  + **MapStruct** hoặc **ModelMapper** là các thư viện phổ biến giúp chuyển đổi giữa Entity và DTO.
  + **MapStruct**:

@Mapper

public interface UserMapper {

UserDTO toDTO(User user);

User toEntity(UserDTO userDTO);

}

* + Với @Mapper, MapStruct sẽ tự động tạo các lớp chuyển đổi khi biên dịch.
* **Dùng Java Reflection**:
  + Java Reflection có thể được dùng để chuyển đổi nhưng sẽ chậm hơn so với MapStruct và ít được tối ưu.

public UserDTO convertToDTO(User user) {

UserDTO dto = new UserDTO();

dto.setName(user.getName());

// Set các field khác...

return dto;

}

**6. Nâng cao**

**6.1 Native Query**

* JPA hỗ trợ native query để thực thi các câu SQL thuần túy khi cần tối ưu truy vấn hoặc xử lý các tác vụ đặc biệt.

java

Copy code

@Query(value = "SELECT \* FROM users WHERE status = ?1", nativeQuery = true)

List<User> findByStatus(String status);

**6.2 Specification và Paging**

* **Specification**: Một cách để tạo các truy vấn động trong Spring JPA.

java

Copy code

public class UserSpecification implements Specification<User> {

private String name;

public Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {

return cb.equal(root.get("name"), name);

}

}

* **Paging**: Spring Data JPA hỗ trợ sẵn với Pageable.

java

Copy code

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Page<User> findAll(Pageable pageable);

}

**6.3 Lấy dữ liệu khi JOIN nhiều bảng với JPA**

* JPA hỗ trợ JOIN qua JPQL hoặc Criteria API.

java

Copy code

@Query("SELECT new com.example.dto.UserOrderDTO(u.name, o.amount) " +

"FROM User u JOIN u.orders o WHERE u.status = :status")

List<UserOrderDTO> fetchUserOrders(@Param("status") String status);

* + Với @Query, bạn có thể chọn lọc chỉ các trường cần thiết khi JOIN các bảng.