06

JPA연관성

01. JPA 연관 관계 매핑

- 데이터베이스 테이블 간의 관계를 엔티티 객체 간의 관계로 표현
- 객체 지향 프로그래밍과 관계형 데이터베이스의 간극을 해소
- 주요 연관 관계:
  - 일대일, 다대일, 일대다, 다대다

02. 일대일 관계 (@OneToOne)

- 한 엔티티가 다른 엔티티와 1:1로 매핑
- 주로 상세 정보를 별도 테이블로 관리할 때 사용
- 예시: 사용자와 사용자 프로필 관계

```
@Entity
public class User {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToOne
    @JoinColumn(name = "profile_id") // 외래 키 지정
        private Profile profile;
}
@Entity
public class Profile {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String bio;
}
```

03. Many-to-One (N:1 관계) : 단방향 관계

- 여러 개의 엔티티가 하나의 엔티티와 매핑
- 예: 게시글과 작성자
  - 여러 게시글이 동일한 작성자를 가질 수 있음.
  - One-to-Many와 반대 방향에서 바라본 관계

```
@Entity
public class Post {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String title;
    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
}

@Entity
public class User {
    @Id @GeneratedValue
    private Long id;
    private String name;
}
```

04. One-to-Many (1:N 관계) : 양방향 관계

- 하나의 엔티티가 여러 개의 다른 엔티티와 매핑
- 예: 사용자와 게시글
  - 한 사용자는 여러 개의 게시글을 가질 수 있음.

```
@Entity
public class User {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "user") // 게시글에서 user 필드와 매핑
        private List<Post> posts = new ArrayList<>(); // 실제 테이블에는 존재하지 않음
}

@Entity
public class Post {
    @Id @GeneratedValue
    private Long id;
    private String title;
    @ManyToOne
    @JoinColumn(name = "user_id") // 외래 키 지정
        private User user;
}
```

04. One-to-Many (1:N 관계) : 양방향 관계

- 서로의 정보를 가지고 있기 때문에 직렬화시 순환참조의 문제가 있음
- @JsonManagedReference : 직렬화에 포함되는 쪽
- @JsonBackReference : 직렬화에 포함되지 않는 쪽
- 양방향이므로 어느쪽이 주인이 되는지 반드시 지정해야 함
  - mappedBy : 주인이 되는 쪽을 지정
  - 비주인이 되는 쪽은 데이터베이스에 영향을 주지 않음
  - 주인이 되는 쪽이 외래키를 관리함

05. Many-to-Many (N:M 관계)

- 여러 개의 엔티티가 서로 여러 개와 매핑
- 예: 학생과 강의
  - 한 학생은 여러 강의를 수강할 수 있고, 한 강의는 여러 학생이 수강할 수 있음
- @ManyToMany를 사용하며, 중간 테이블이 필요
  - Jpa가 자동으로 생성
  - 중간테이블에 양쪽 테이블과 연결되는 필드 이외에 필드가 필요하다면 직접 생성할 수도 있음

05. Many-to-Many (N:M 관계)

```java
@Entity
public class Student {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String name;
    @ManyToMany
    @JoinTable(
            name = "student_course",
            joinColumns = @JoinColumn(name = "student_id"),
            inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private List<Course> courses = new ArrayList<>();
}
```

```java
@Entity
public class Course {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private List<Student> students = new ArrayList<>();
}
```

06. 연관성 매핑 시 주의점

- **mappedBy 속성**
  - 관계의 주인을 정의
  - 주인은 외래 키를 관리하며, 반대쪽은 읽기 전용
  - 일반적으로 @OneToMany, @ManyToMany에서 사용
- **Fetch Type (지연 로딩 vs 즉시 로딩)**
  - Eager Fetch (즉시 로딩)
    - 연관된 엔티티를 즉시 조회
    - 기본값: @OneToOne, @ManyToOne
  - Lazy Fetch (지연 로딩)
    - 연관된 엔티티를 실제로 사용할 때 조회
    - 기본값: @OneToMany, @ManyToMany

```
@OneToMany(fetch = FetchType.LAZY)
private List<Post> posts;
```

07. 연관성 매핑 활용방법

- 연관 관계의 주인 명확히 설정
  - mappedBy를 사용해 주인을 설정하고, 외래 키를 관리
- 지연 로딩(Lazy)을 기본으로 설정
  - 성능 최적화를 위해 필요할 때만 데이터를 로드
- Cascade 옵션 활용
  - 연관된 엔티티를 자동으로 저장, 삭제하려면 cascade 속성을 설정

```
@OneToMany(cascade = CascadeType.ALL,
                        orphanRemoval = true)
private List<Post> posts;
```