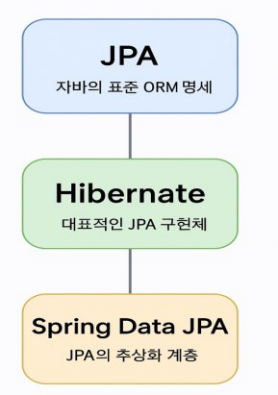
[실습 1] Spring JPA 로 SpringBootLab04

Spring Boot JPA(또는 Hibernate JPA) ?

1. JPA (Java Persistence API)



- 자바의 **표준 ORM 명세**
- DB 와 엔티티를 매핑하고, 영속성 관리(저장/조회 등)를 위한 API 집합
- 특정 구현체에 의존하지 않음 (추상화)
- 핵심 구성: `EntityManager`, `@Entity`, `@Id`, `@Table`, `@Column` 등

---

2. Hibernate

- JPA 의 **대표적 구현체**
- JPA 가 "이렇게 해야 한다"고 정의하면, Hibernate 가 "그렇게 실행"함
- 자체 기능도 많지만, 스프링 부트에서는 주로 JPA 표준 위주로 사용

---

3. Spring Data JPA

- Spring 의 **JPA 추상화 계층**
- `JpaRepository` 상속만으로 CRUD 기능 자동 제공
- 복잡한 쿼리도 메서드명 규칙이나 `@Query` 로 쉽게 처리
- **직접 JPA 를 구현하지는 않고**, Hibernate 등 JPA 구현체 위에서 동작

JPA 학습 단계

단계	내용
[1] 엔티티 선언	@Entity, @Id, @GeneratedValue 로 엔티티와 기본키 정의
[2] 관계 매핑	@ManyToOne, @OneToMany 등 관계 설정 (Fetch: LAZY, EAGER)
[3] 영속성 컨텍스트	1 차 캐시, Dirty Checking 으로 엔티티 상태 관리
[4] Repository	JpaRepository 상속 및 메서드명 기반 쿼리 자동 생성
[5] JPQL	@Query 로 복잡한 쿼리 작성
[6] 테스트	@DataJpaTest, @Transactional 로 Repository 단위 테스트

JPA 주요 애너테이션

◇ 엔티티 & 식별자

@	설명
@Entity	JPA 에서 관리할 엔티티 클래스임을 명시
@Id	기본 키(primary key) 필드 지정
@GeneratedValue	기본 키 자동 생성 전략 설정 (AUTO, IDENTITY 등)

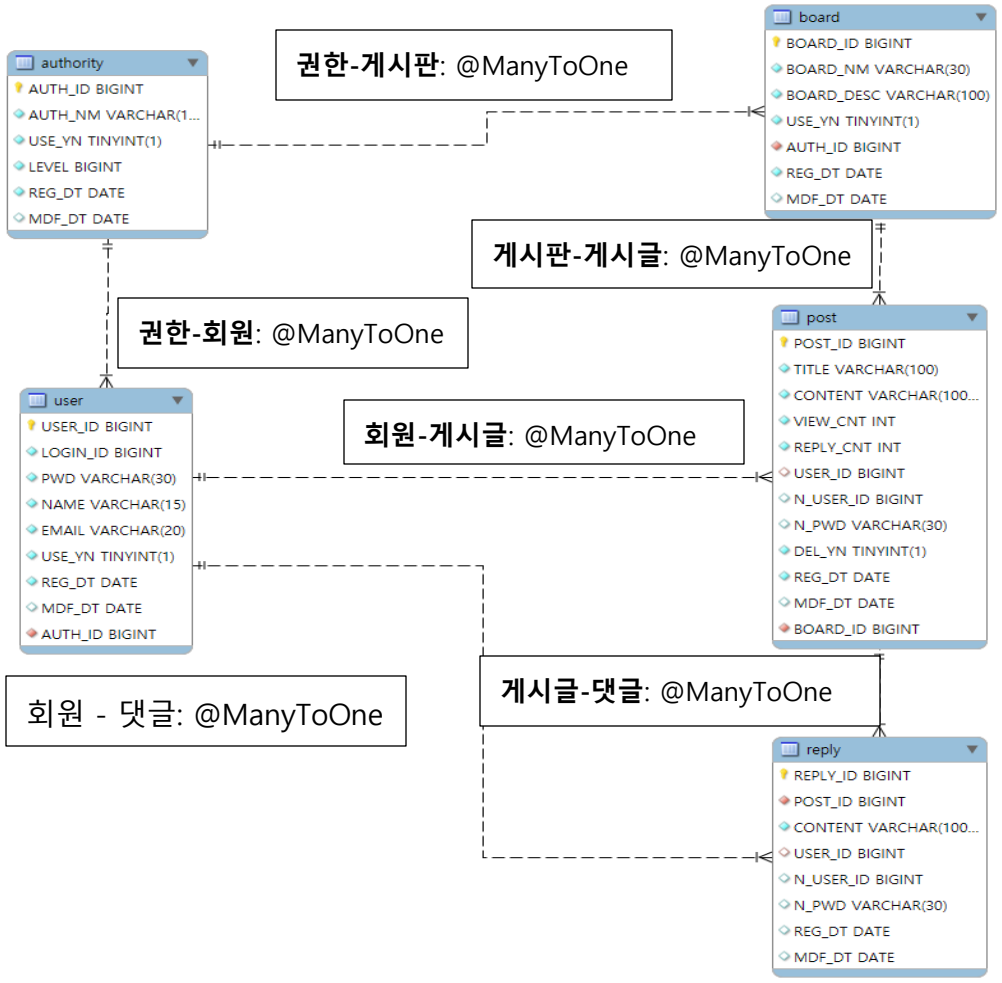
칼럼 매핑

@	설명
@Column	DB 컬럼 속성 지정 (길이, 이름 등)
@Transient	DB 와 매핑하지 않을 필드 (임시 데이터용)
@Lob	BLOB/CLOB 대용량 데이터 매핑
@Enumerated	enum 매핑 방식 (ORDINAL, STRING) 지정

연관 관계 매핑

단방향/양방향 관계

@	설명	샘플 예제 (간단하게)
@ManyToOne	다대일 관계 (ex. 여러 사원이 한 부서에 소속)	@ManyToOne private Dept dept;
@OneToMany	일대다 관계 (ex. 한 부서에 여러 사원)	@OneToMany(mappedBy="dept") private List<Emp> emps;
@OneToOne	일대일 관계 (ex. 사원과 사원정보가 1:1)	@OneToOne private EmpInfo empInfo;
@ManyToMany	다대다 관계 (ex. 학생-강의)	@ManyToMany private List<Course> courses;



테이블 명세서

AUTHORITY

컬럼명	타입	NULL 허용	설명
AUTH_ID	BIGINT(10)	NO	권한 ID (PK)
AUTH_NM	VARCHAR(15)	NO	권한명
USE_YN	TINYINT(1)	NO	사용 여부
LEVEL	BIGINT(10)	NO	권한 레벨
REG_DT	DATE	NO	등록일
MDF_DT	DATE	YES	수정일

USER

컬럼명	타입	NULL 허용	설명
USER_ID	BIGINT(10)	NO	회원 ID (PK)
LOGIN_ID	BIGINT(10)	NO	로그인 ID
PWD	VARCHAR(30)	NO	비밀번호
NAME	VARCHAR(15)	NO	이름
EMAIL	VARCHAR(20)	NO	이메일
USE_YN	TINYINT(1)	NO	사용 여부
REG_DT	DATE	NO	등록일
MDF_DT	DATE	YES	수정일
AUTH_ID	BIGINT(10)	NO	권한 ID (FK → AUTHORITY.AUTH_ID)

BOARD

컬럼명	타입	NULL 허용	설명
BOARD_ID	BIGINT(10)	NO	게시판 ID (PK)
BOARD_NM	VARCHAR(30)	NO	게시판 이름
BOARD_DESC	VARCHAR(100)	NO	게시판 설명
USE_YN	TINYINT(1)	NO	사용 여부

AUTH_ID	BIGINT(10)	NO	권한 ID (FK → AUTHORITY.AUTH_ID)
REG_DT	DATE	NO	등록일
MDF_DT	DATE	YES	수정일

POST

컬럼명	타입	NULL 허용	설명
POST_ID	BIGINT(10)	NO	게시글 ID (PK)
TITLE	VARCHAR(100)	NO	제목
CONTENT	VARCHAR(1000)	NO	내용
VIEW_CNT	INT(100)	NO	조회수
REPLY_CNT	INT(100)	NO	댓글수
USER_ID	BIGINT(10)	YES	회원 ID (FK → USER.USER_ID)
N_USER_ID	BIGINT(10)	YES	비회원 ID (nullable)
N_PWD	VARCHAR(30)	YES	비회원 비밀번호 (nullable)
DEL_YN	TINYINT(1)	NO	삭제 여부
REG_DT	DATE	NO	등록일
MDF_DT	DATE	YES	수정일
BOARD_ID	BIGINT(10)	NO	게시판 ID (FK → BOARD.BOARD_ID)

REPLY

컬럼명	타입	NULL 허용	설명
REPLY_ID	BIGINT(10)	NO	댓글 ID (PK)
POST_ID	BIGINT(10)	NO	게시글 ID (FK → POST.POST_ID)
CONTENT	VARCHAR(1000)	NO	댓글 내용
USER_ID	BIGINT(10)	YES	회원 ID (FK → USER.USER_ID)
N_USER_ID	BIGINT(10)	YES	비회원 ID (nullable)
N_PWD	VARCHAR(30)	YES	비회원 비밀번호 (nullable)
REG_DT	DATE	YES	등록일
MDF_DT	DATE	YES	수정일

Fetch 전략

속성	설명
fetch=LAZY	실제로 사용할 때만 로딩 (성능 최적화, 기본값)
fetch=EAGER	즉시 로딩 (N+1 문제 주의) @ManyToOne, @OneToOne 관계의 **기본 Fetch 전략**

N+1 문제란?

- 지연 로딩(FetchType.LAZY)에서 **1 개의 쿼리(N)** 로 엔티티를 조회한 뒤, 각 엔티티의 연관된 데이터를 조회할 때마다 추가 쿼리가 발생.
- 결과적으로 **총 N+1 개의 쿼리**가 실행됨!
- 예)

```
1) SELECT * FROM member;  -- Member N 개 조회
2) SELECT * FROM order WHERE member_id=?;  -- N 번 실행
```

해결 전략

해결 전략	설명	특징
페치 조인 (Fetch Join)	JPQL 의 JOIN FETCH 키워드로 연관 엔티티를 **즉시 로딩**처럼 한 번에 가져옴.	- 가장 효과적 - JPQL 에 직접 쿼리 작성 필요
@EntityGraph	Spring Data JPA 에서 제공. 메서드 호출 시 **어노테이션으로 페치조인**을 선언.	- 쿼리 메서드 그대로 사용 가능 - 동적 사용에 유리
Batch Size 설정	@BatchSize 또는 Hibernate hibernate.default_batch_fetch_size 를 설정. LAZY 로딩 시에도 **IN 절로 묶어 조회**.	- 컬렉션별로 설정 가능 - N+1 문제는 줄지만 쿼리 수는 완전히 1 로 줄어들지 않음

Cascade 옵션

옵션	설명
CascadeType.ALL	모든 영속성 전이 (SAVE, DELETE 등)
PERSIST, REMOVE 등	개별 동작만 전이

영속성 컨텍스트

용어	설명
영속성 컨텍스트	엔티티를 영속 상태로 관리 (1 차 캐시 역할)
merge()	Detached 상태 엔티티를 다시 영속성 컨텍스트에 등록
Dirty Checking	트랜잭션 종료 시 변경된 필드만 업데이트 쿼리 생성

Repository 인터페이스

기본 인터페이스

인터페이스	설명
JpaRepository<T, ID>	CRUD 기본 메서드 제공 (findAll, save, delete 등)
CrudRepository	CRUD 만 제공 (더 단순)
PagingAndSortingRepository	페이징 및 정렬 기능 추가

메서드 이름 기반 쿼리: 메서드명으로 자동 JPQL 쿼리 생성!

Spring Data JPA 의 메서드명 기반 쿼리 자동 생성은 단일 엔티티(=한 테이블) 에 대한 조건만 지원

```
List<Emp> findByDeptno(int deptno);
List<Emp> findByEnameContaining(String keyword);
```

기본원리

findBy + [엔티티 속성명] -> 규칙

규칙	설명
findBy…	조건으로 찾기
And / Or	조건 추가 (AND / OR)
Between	값 사이
LessThan	작다 (<)
GreaterThan	크다 (>)
Like	LIKE 검색
In	IN 절 검색
OrderBy…Asc	오름차순 정렬
OrderBy…Desc	내림차순 정렬

Ex)이름으로 오름차순 조회

List<Member> findByNameOrderByIdAsc(String name);

SELECT * FROM member WHERE name = ? ORDER BY id ASC

JPQL & @Query : JPA 의 객체지향 쿼리 언어

```
@Query("SELECT e FROM Emp e WHERE e.dept.deptno = :deptno")
List<Emp> findByDeptno(@Param("deptno") int deptno);
```

테스트 & 트랜잭션

@	설명
@DataJpaTest	JPA 레포지토리 테스트에 특화된 슬라이스 테스트
@Transactional	테스트에서 자동 롤백 처리
@Rollback(false)	롤백을 안 하고 실제 DB 반영

[실습] SpringLab04 Emp-Dept 구조 (1:N) JPA

디렉토리 구조

```
SpringBootLab04
  ├─── src/main/java
  |     └─── com.sec01
  |          ├─── Dept.java
  |          ├─── DeptRepository.java
  |          ├─── Emp.java
  |          ├─── EmpRepository.java
  |          ├─── EmpService.java
  |          ├─── EmpController.java
  |          └─── SpringBootLab04Application.java
  ├─── src/main/resources
  |     ├─── application.yml (또는 application.properties)
   ├─── src/main/resources/templates/
            └─── emp/
                      ├─── list.html , new.html ,detail.html , edit.html
```

Application.yml

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/my_emp
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update # 또는 create, create-drop, none
    show-sql: true # JPA 가 생성하는 SQL 을 콘솔에 출력
    properties:
      hibernate:
        format_sql: true # SQL 포맷팅
logging:
  level:
    org:
      hibernate:
        SQL: DEBUG # 실행되는 SQL 쿼리 로깅
        type:
          descriptor:
            sql: TRACE # SQL 파라미터 로깅
```

Entity

```java
@Entity @NoArgsConstructor
@AllArgsConstructor
public @Data class Emp {
    @Id
    private int empno;
    private String ename;
    private String job;
    private int sal;

    @ManyToOne
    @JoinColumn(name = "deptno")
    private Dept dept;
}
```

```java
@Entity
@NoArgsConstructor @AllArgsConstructor
public @Data class Dept {
    @Id
    private int deptno;
    private String dname;
    private String loc;

    @OneToMany(mappedBy = "dept")
    private List<Emp> emps;
}
```

Repository

Entity	Repository 이름
Emp	EmpRepository
Dept	DeptRepository

```java
@Repository
public interface EmpRepository extends JpaRepository<Emp, Integer> { }
```

```java
@Repository
public interface DeptRepository extends JpaRepository<Dept, Integer> {
}
```

EmpService

메서드명	매개변수	반환형	설명
findAll()	-	List<Emp>	모든 사원 조회
findById()	int empno	Emp	사번으로 사원 상세 조회
save()	Emp emp	void	사원 저장 (등록 & 수정)
delete()	int empno	void	사원 삭제

DeptService

메서드명	매개변수	반환형	설명
findAll()	-	List<Dept>	모든 부서 조회
findById()	int deptno	Dept	부서번호로 부서 상세 조회

Cotroller

번호	경로 (URL)	메서드	기능 설명	템플릿 (뷰)
1	/emp	GET	사원 목록 및 부서 목록 조회	emp/list.html
2	/emp/new	GET	사원 등록 화면 표시	emp/new.html
3	/emp	POST	사원 등록 처리	(리다이렉트)
4	/emp/{empno}	GET	사원 상세 정보 조회	emp/detail.html
5	/emp/{empno}/edit	GET	사원 수정 화면 표시	emp/edit.html
6	/emp/{empno}/edit	POST	사원 수정 처리	(리다이렉트)
7	/emp/{empno}/delete	POST	사원 삭제 처리	(리다이렉트)
8	/emp/list	GET	사원 목록 전용 경로 (추가됨)	emp/list.html

CRUD 실행한다.

[추가 실습] 메서드명 기반 쿼리를 추가해서 확인 해보자.

✅ **쿼리 메서드 → 서비스 → 컨트롤러 → 뷰 (list.html)**

✅ 결과는 Thymeleaf 템플릿에서 th:each 로 반복 출력!

EmpRepository 쿼리 메서드

메서드명	설명	결과 SQL 예시
findByEname(String ename)	이름으로 사원 검색	SELECT * FROM emp WHERE ename = ?
findByDeptDeptno(int deptno)	부서번호로 사원 검색	SELECT * FROM emp WHERE deptno = ?
findBySalGreaterThanEqual(int sal)	급여가 이상인 사원 검색	SELECT * FROM emp WHERE sal >= ?
findByEnameContaining(String keyword)	이름에 특정 키워드가 포함된 사원 검색	SELECT * FROM emp WHERE ename LIKE %keyword%

DeptRepository 쿼리 메서드

메서드명	설명	결과 SQL 예시
findByDname(String dname)	부서명으로 부서 검색	SELECT * FROM dept WHERE dname = ?
findByDnameContaining(String keyword)	부서명에 특정 키워드가 포함된 부서 검색	SELECT * FROM dept WHERE dname LIKE %keyword%