[실습 1] Spring JPA 로 SpringBootLab5

Spring Boot JPA(또는 Hibernate JPA)?

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

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;

[실습] SpringLab05 Emp-Dept 구조 (1:N) JPA

1) 디렉토리 구조

```
SpringLab05 /
└──── src/
    └──── main/
        ├──── java/com/sec01/
        |    ├──── Sec01Application.java
        |    ├──── controller/
        |    |    └──── EmpDeptController.java
        |    ├──── service/
        |    |    └──── EmpDeptService.java
        |    ├──── repository/
        |    |    ├──── EmpRepository.java
        |    |    ├──── DeptRepository.java
        |    |    └──── EmpDeptRepository.java
        |    ├──── entity/
        |    |    ├──── Emp.java
        |    |    └──── Dept.java
        |    └──── dto/
        |         └──── EmpDeptDto.java
        └──── resources/
            ├──── templates/
            |    └──── emp-info.html
            └──── application.properties
```

2) 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
```

3) @Query

```
@Query("JPQL 또는 네이티브 쿼리")
반환타입 메서드명(파라미터);
```

차이

구분	JPQL	네이티브 쿼리
기준	엔티티, 필드	테이블, 컬럼
DB 독립성	좋음 (DBMS 상관없이 변환됨)	DBMS 에 따라 달라질 수 있음
복잡한 쿼리 처리	복잡한 SQL 은 힘들다	가능 (DB 전용 함수, 뷰 등 활용)
성능 최적화	JPA 가 최적화	직접 SQL 튜닝 가능

주요속성

속성명	설명	기본값
value	실행할 쿼리문 (JPQL 또는 SQL)	필수
nativeQuery	쿼리를 **네이티브 쿼리(SQL)**로 처리할지 여부	false (JPQL 로 처리)
countQuery	페이징 처리 시, 결과 수를 반환하는 카운트 쿼리 (Pageable/페이징)	생략 가능 (자동 생성)

주요 클래스

클래스명	풀네임
Pageable	org.springframework.data.domain.Pageable
Page	org.springframework.data.domain.Page
PageRequest	org.springframework.data.domain.PageRequest
Sort	org.springframework.data.domain.Sort

Case 1: JPQL :파라미터명 으로 바인딩 @Param 으로 연결

① 엔티티 기반으로 동작 (테이블, 컬럼명이 아니라 클래스, 필드를 기준으로 함)

② DB 독립적 : JPA 구현체가 SQL 로 변환해줌

③ 장점 : DB 변경해도 쿼리 수정 거의 없음

```
@Query("SELECT e FROM Emp e WHERE e.sal > : minSal")
List<Emp> findEmpWithMinSalary(@Param("minSal") int minSal);
```

Case 2 : nativeQuery =true 로 네이티브 쿼리 지정

① DB 에서 바로 실행되는 쿼리문 (SQL)

② 장점: 복잡한 SQL 쿼리, DB 전용 기능, 뷰(view) 같은 특수한 쿼리 처리 가능

③ 단점: DB 종속적 (DBMS 바꾸면 쿼리 다시 작성해야 할 수 있음)

```
@Query(value = "SELECT * FROM EMP WHERE SAL > :minSal", nativeQuery =
true)
List<Emp> findEmpNative(@Param("minSal") int minSal);
```

Case 3: 페이징 처리

```
@Query(value = "SELECT e FROM Emp e",
      countQuery = "SELECT count(e) FROM Emp e")
Page<Emp> findAllWithPaging(Pageable pageable);
```

4) Pageable 이란?

① **페이지 요청 정보**를 담은 객체
② 보통 **page 번호, size, 정렬** 정보를 담음
③ 인터페이스 형태로 제공
④ **컨트롤러**나 **서비스**에서 사용자가 원하는 페이징 정보를 받아오는 역할

```
Pageable pageable = PageRequest.of(0, 5); // 0 번 페이지, 5 개씩 조회
```

```
//정렬기준
Pageable pageable = PageRequest.of(0, 5, Sort.by("ename").descending());
```

주요 메소드

메서드	설명
getPageNumber()	현재 페이지 번호 (0 부터 시작)
getPageSize()	한 페이지당 데이터 개수
getOffset()	전체 데이터 중에서의 시작 위치
getSort()	정렬 정보

5) Page<T>란?

① **페이지 결과 전체를 담는 객체**
② JPA 쿼리 실행 결과를 담음
③ 내부적으로는 List<T> + 전체 페이지 정보

주요 메소드

메서드	설명
getContent()	현재 페이지의 데이터 리스트 반환
getTotalPages()	전체 페이지 개수 반환
getTotalElements()	전체 요소 수 반환
getNumber()	현재 페이지 번호 (0 부터 시작)
getSize()	한 페이지당 몇 개씩 가져오는지 반환
hasNext()	다음 페이지가 있는지 여부
hasPrevious()	이전 페이지가 있는지 여부

구현단계

[1] 리포지토리 단계

EmpDeptRepository 에서 @Query 어노테이션을 사용해, Emp 와 Dept 를 조인한 결과를 EmpDeptDto 로 매핑해 반환하는 쿼리를 작성한다. **Pageable** 파라미터를 통해 페이징 처리 기능을 지원하도록 한다.

```
@Query("SELECT e FROM Emp e")
Page<Emp> findAllWithPaging(Pageable pageable);
```

[2] 서비스 단계

EmpDeptService 의 getEmpDeptPage() 메서드는 **Pageable** 객체를 받아,

EmpDeptRepository 의 findEmpDeptPage() 메서드를 호출하여 **페이징된 EmpDeptDto 결과**를 그대로 반환한다.

```
public Page<Emp> getPagedEmps(Pageable pageable) {
    return empRepository.findAllWithPaging(pageable);
}
```

```
@GetMapping("/emp-list")
public String empList(@RequestParam(defaultValue = "0") int page,
                      @RequestParam(defaultValue = "5") int size,
                      Model model) {

     Pageable pageable = PageRequest.of(page, size);
    Page<Emp> empPage = service.getPagedEmps(pageable);
    model.addAttribute("empPage", empPage);
    return "emp-list";
}
```

[3] 뷰(템플릿) 단계

Thymeleaf 뷰에서는 페이징 결과(empPage)를 기반으로 이전/다음 링크와 현재 페이지 번호/전체 페이지 수를 동적으로 표시한다. 이때, hasPrevious(), hasNext() 메서드를 사용해 페이지 존재 여부를 확인하고, number 와 totalPages 로 페이지 번호를 출력한다.

```
<a th:href="@{|/emp-info?page=${empPage.number - 1}|}"
   th:if="${empPage.hasPrevious()}">이전</a>
<span th:text="${empPage.number + 1}"></span> / <span th:text="${empPage.totalPages}"></span>
<a th:href="@{|/emp-info?page=${empPage.number + 1}|}"
   th:if="${empPage.hasNext()}">다음</a>
```

Entity

```
@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;
}
```

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

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

Dto

```
@Data
@AllArgsConstructor
public class EmpDeptDto {
    private String ename;
    private int sal;
    private String dname;
}
```

Repository

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

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

EmpDeptRepository

```
@Repository
public interface EmpDeptRepository extends JpaRepository<Emp,
Integer> {

    @Query("SELECT new com.sec01.dto.EmpDeptDto(e.ename, e.sal,
d.dname) " +
            "FROM Emp e JOIN e.dept d")
    List<EmpDeptDto> findEmpDeptInfo();

    @Query(value = "SELECT e.ENAME, e.SAL, d.DNAME " +
      "FROM EMP e JOIN DEPT d ON e.DEPTNO = d.DEPTNO",
       nativeQuery = true)
    List<Object[]> findEmpDeptNative();


  @Query("SELECT new com.sec01.dto.EmpDeptDto(e.ename, e.sal,
   d.dname) " +  "FROM Emp e JOIN e.dept d")
    Page<EmpDeptDto> findEmpDeptPage(Pageable pageable);
}
```

EmpDeptService

메서드명	반환타입	설명
getEmpDeptDtos()	List<EmpDeptDto>	Emp 와 Dept 를 조인해서 DTO 로 가져오는 메서드
getAllEmps()	List<Emp>	Emp 엔티티 전부 조회
getAllDepts()	List<Dept>	Dept 엔티티 전부 조회
getEmpDeptPage()	Page<EmpDeptDto>	Emp 와 Dept 조인 결과를 페이지 단위로 조회 (페이징 처리)

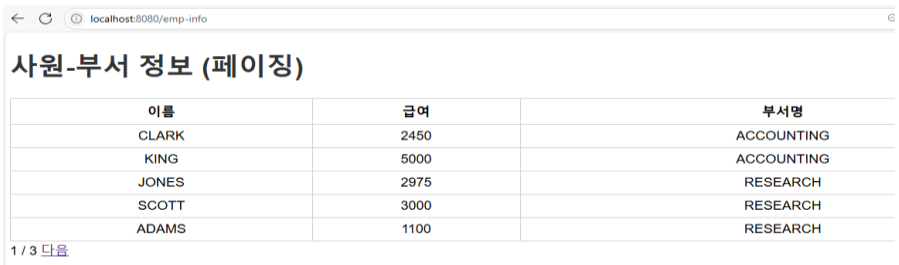
Cotroller

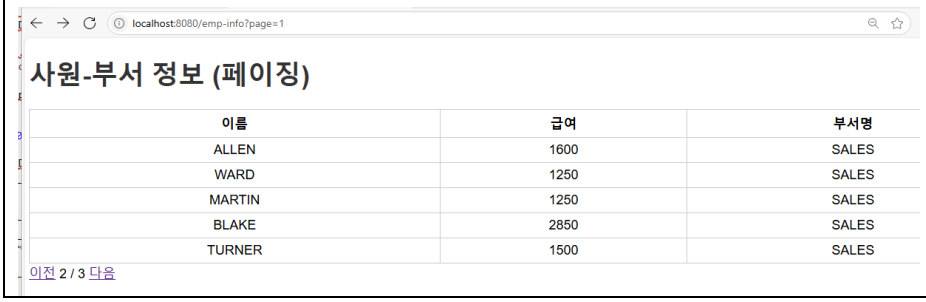
경로 (URL)	컨트롤러 메서드명	HTTP 메서드	기능 설명	템플릿 (뷰)
/emp-info	showEmpDeptInfo()	GET	Emp 와 Dept 를 조인한 결과를 페이징 처리해서 뷰로 전달	emp-info.html
/emps	showAllEmps()	GET	Emp 테이블의 전체 사원 목록을 뷰로 전달	emps.html
/depts	showAllDepts()	GET	Dept 테이블의 전체 부서 목록을 뷰로 전달	depts.html

각 url 실행

사원 목록

번호	이름	직무	급여	부서명
7499	ALLEN	SALESMAN	1600	SALES
7521	WARD	SALESMAN	1250	SALES
7566	JONES	MANAGER	2975	RESEARCH
7654	MARTIN	SALESMAN	1250	SALES
7698	BLAKE	MANAGER	2850	SALES
7782	CLARK	MANAGER	2450	ACCOUNTING
7788	SCOTT	ANALYST	3000	RESEARCH
7839	KING	PRESIDENT	5000	ACCOUNTING
7844	TURNER	SALESMAN	1500	SALES
7876	ADAMS	CLERK	1100	RESEARCH
7900	JAMES	CLERK	950	SALES





[추가 실습 01] Pageable 에 Sort 정보 추가를 해보자

Pageable 객체에 정렬 정보를 전달한다.

PageRequest.of(page, size, Sort.by(컬럼명))

Controller

```
@GetMapping("/emp-info")
public String showEmpDeptInfo(Model model,
                              @RequestParam(defaultValue = "0") int page,
                              @RequestParam(defaultValue = "5") int size,
                              @RequestParam(defaultValue = "ename") String sortBy,
                              @RequestParam(defaultValue = "asc") String direction) {
    Sort sort = direction.equalsIgnoreCase("asc") ? Sort.by(sortBy).ascending()
                                                  : Sort.by(sortBy).descending();
    Pageable pageable = PageRequest.of(page, size, sort);
    Page<EmpDeptDto> empPage = service.getEmpDeptPage(pageable);
    model.addAttribute("empPage", empPage);
    return "emp-info";
}
```

View

```
<div>
    <a th:href="@{|/emp-info?page=0&sortBy=ename&direction=asc|}">이름
오름차순</a>
    <a th:href="@{|/emp-info?page=0&sortBy=ename&direction=desc|}">이름
내림차순</a>
    <a th:href="@{|/emp-info?page=0&sortBy=sal&direction=asc|}">급여
오름차순</a>
</div>
```

[추가실습 02] 1 차 캐시, Dirty Checking, FetchType(LAZY/EAGER)

TestFile : EmpPersistenceContextTest.java

[1] 1 차 캐시 테스트 : EntityManager 가 관리하는 엔티티 저장소

- 트랜잭션 범위 안에서 **DB 에서 조회한 엔티티**를 **메모리**에 **캐싱한다.**

```
@SpringBootTest
public class EmpPersistenceContextTest {

    @PersistenceContext
        EntityManager entityManager;

    @Transactional
    public void testFirstLevelCache() {
        Emp emp1 = entityManager.find(Emp.class, 7369);  //-> 캐시보관
        Emp emp2 = entityManager.find(Emp.class, 7369); // -> emp1 리턴
        System.out.println(emp1 == emp2); // true
    }
}
```

[2] Dirty Checking 테스트

```
@Test
    @Transactional
    public void testDirtyChecking() {
        Emp emp = entityManager.find(Emp.class, 7369);
        System.out.println("기존 급여: " + emp.getSal());

        emp.setSal(emp.getSal() + 100); // 급여 변경 (Dirty Checking 발생)

        System.out.println("변경된 급여: " + emp.getSal());
        // 트랜잭션 커밋 시점에 UPDATE 쿼리가 나감
    }
```

[3] 관계 매핑 (LAZY / EAGER) 테스트

```
@Test
    @Transactional
    public void testFetchType() {
        Emp emp = entityManager.find(Emp.class, 7369);
        System.out.println("사원이름: " + emp.getEname());

        // LAZY 관계라면 아래에서 쿼리 발생
        System.out.println("부서이름: " + emp.getDept().getDname());
    }
```