Spring Boot Application

스프링 부트 응용

JPA API 이해하기

한국기술교육대학교
온라인평생교육원

학습내용

- EntityManager와 영속 컨텍스트
- 영속 컨텍스트와 엔티티 상태
- 영속 컨텍스트와 SQL 저장소

학습목표

- 영속 컨텍스트의 개념을 이해하고 EntityManager와의 관계를 설명할 수 있다.
- EntityManager가 제공하는 API를 이용하여 엔티티 상태를 관리할 수 있다.
- 영속 컨텍스트에 있는 1차 캐시와 SQL 저장소의 동작 원리를 설명할 수 있다.

EntityManager와 영속 컨텍스트

1 EntityManagerFactory와 EntityManager

〈1〉 EntityManager 획득

- JPA에서 엔티티 객체를 이용하여 CRUD 기능을 사용하려면 반드시 EntityManager 객체를 획득해야 함

1 EntityManagerFactory와 EntityManager

〈1〉 EntityManager 획득

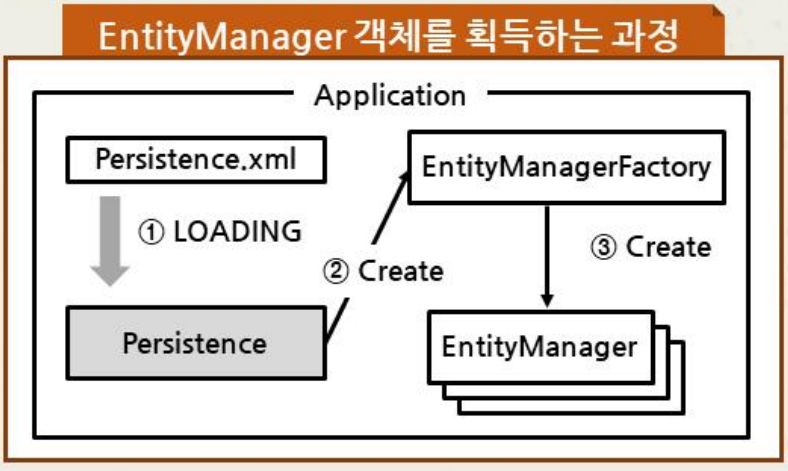
- EntityManager는 EntityManagerFactory로부터 얻을 수 있음
- 이때 persistence.xml 파일에 설정된 영속성 유닛 정보가 로딩됨

```
// EntityManager 생성
EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest");
EntityManager em = emf.createEntityManager();
```

EntityManager와 영속 컨텍스트

1 EntityManagerFactory와 EntityManager

〈1〉 EntityManager 획득



1 EntityManagerFactory와 EntityManager

〈2〉 EntityManager 메소드

- EntityManager가 제공하는 메소드를 이용하여 CRUD 기능을 구현할 수 있음

EntityManager와 영속 컨텍스트

1 EntityManagerFactory와 EntityManager

〈2〉 EntityManager 메소드

- EntityManager가 제공하는 메소드의 종류와 기능

메소드	기능 설명
persist(Object entity)	엔티티를 영속화 함(INSERT)
merge(Object entity)	준영속 상태의 엔티티를 영속화 함(UPDATE)
remove(Object entity)	영속 상태의 엔티티를 제거함(DELETE)
find(Class<T> entityClass, Object primaryKey)	하나의 엔티티를 검색함(SELECT ONE)
createQuery(String jpql, Class<T> resultClass)	JPQL에 해당하는 엔티티 목록을 검색함(SELECT LIST)

1 EntityManagerFactory와 EntityManager

〈2〉 EntityManager 메소드

- 등록/수정/삭제 작업은 반드시 트랜잭션 안에서 처리되어야 실제 SQL이 생성되고 처리됨

영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

〈1〉 영속(Persistence Context) 컨텍스트

- EntityManager를 생성할 때 자동으로 만들어짐
- 엔티티 객체들을 관리하는 일종의 컨테이너

1 영속 컨텍스트와 엔티티 상태

〈1〉 영속(Persistence Context) 컨텍스트

- 영속 컨텍스트는 EntityManager를 통해서만 접근할 수 있음
- 영속 컨텍스트에 등록된 엔티티는 EntityManager가 제공하는 메소드를 통해 관리됨

영속 컨텍스트와 엔티티 상태

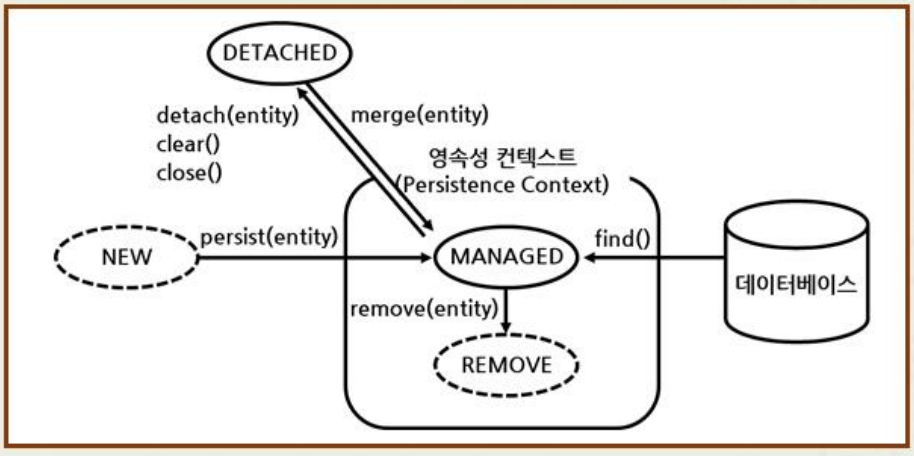
1 영속 컨텍스트와 엔티티 상태

〈2〉 영속 컨텍스트의 엔티티 상태

상태	설명
비영속	엔티티가 영속 컨텍스트와 전혀 무관한 상태
영속	엔티티가 영속 컨텍스트에 저장된 상태
준영속	엔티티가 한번 영속 컨텍스트에 저장되었다가 분리된 상태
삭제	엔티티가 영속 컨텍스트에서 삭제된 상태

1 영속 컨텍스트와 엔티티 상태

〈2〉 영속 컨텍스트의 엔티티 상태

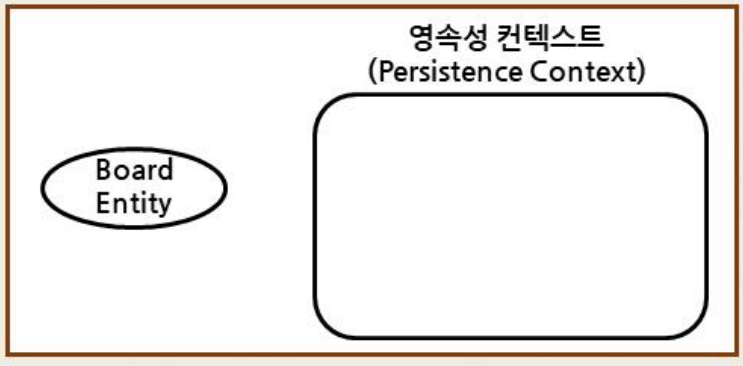


영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

〈3〉 비영속 상태(NEW)

- 엔티티 객체를 생성만 했을 뿐, 아직 엔티티를 영속 컨텍스트에 저장하지 않은 단순한 자바 객체임



1 영속 컨텍스트와 엔티티 상태

〈4〉 영속 상태(MANAGED)

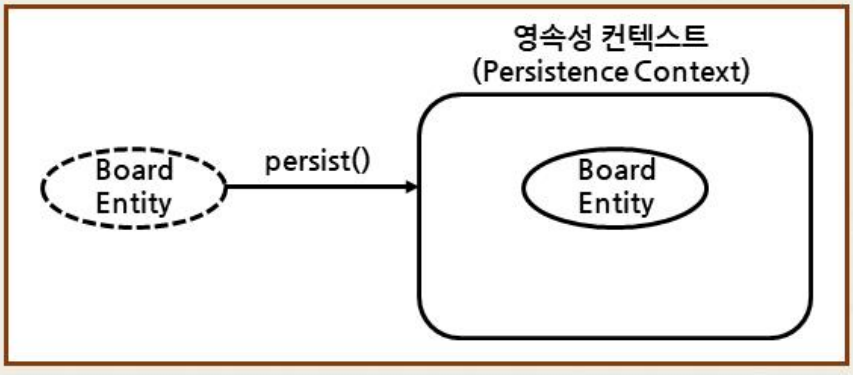
- EntityManager를 통해 엔티티가 영속 컨텍스트에 저장된 상태
- 엔티티를 영속 상태로 만들기 위해서 EntityManager.persist() 메소드를 사용함

영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

④ 영속 상태(MANAGED)

- 엔티티가 영속상태에 있다고 해서 바로 INSERT가 실행되는 것은 아님



1 영속 컨텍스트와 엔티티 상태

④ 영속 상태(MANAGED)

- EntityManager.find 메소드를 통해서도 엔티티를 영속 상태로 만들 수 있음

영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

④ 영속 상태(MANAGED)

- EntityManager.find 메소드

상세 조회하는 엔티티가 영속 컨텍스트에 있을 경우

1 영속 컨텍스트와 엔티티 상태

④ 영속 상태(MANAGED)

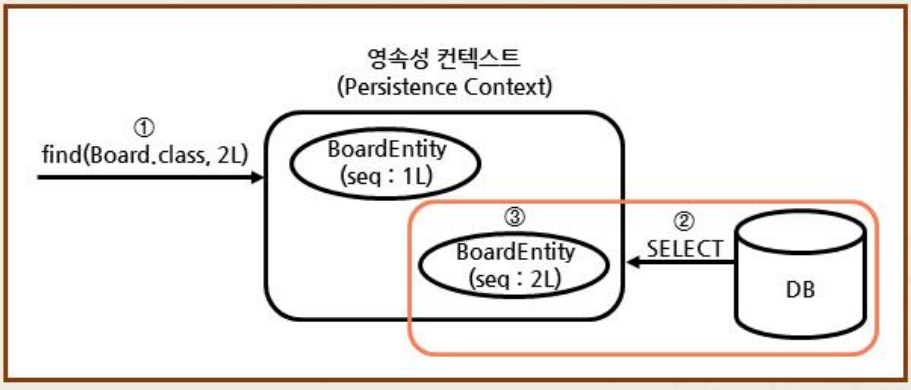
- EntityManager.find 메소드

상세 조회하는 엔티티가 영속 컨텍스트에 없을 경우

영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

④ 영속 상태(MANAGED)



1 영속 컨텍스트와 엔티티 상태

⑤ 준영속 상태(DETATCHED)

- 한번 영속성 컨텍스트에 들어간 엔티티가 어떤 이유에서 영속 컨텍스트에서 벗어난 상태
- 엔티티가 영속성 컨텍스트에서 벗어났기 때문에 값을 수정해도 데이터베이스에 아무런 영향이 없음

영속 컨텍스트와 엔티티 상태

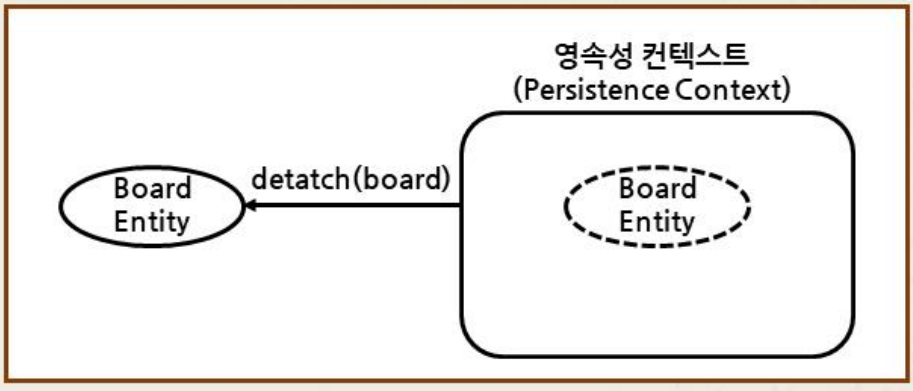
1 영속 컨텍스트와 엔티티 상태

⑤ 준영속 상태(DETATCHED)

- 준영속 상태의 엔티티는 메모리에서 완전히 사라진 것이 아님
- EntityManager.merge 메소드를 통해 다시 영속 상태로 전환될 수 있음

1 영속 컨텍스트와 엔티티 상태

⑤ 준영속 상태(DETATCHED)



영속 컨텍스트와 엔티티 상태

1 영속 컨텍스트와 엔티티 상태

〈5〉 준영속 상태(DETATCHED)

- 영속 상태의 엔티티가 준영속 상태로 전환되는 경우

메소드	설명
detach(Object entity)	특정 엔티티만 준영속 상태로 전환함
clear()	• 영속성 컨텍스트를 초기화함 • 이때 영속 컨텍스트가 관리하던 모든 엔티티들을 준영속 상태로 전환함
close()	• 영속 컨텍스트를 종료함 • 영속 컨텍스는 종료되기 직전 자신이 관리하던 엔티티들을 모두 준영속 상태로 전환함

1 영속 컨텍스트와 엔티티 상태

〈6〉 삭제 상태(REMOVED)

- 엔티티가 영속성 컨텍스트에서도 제거되고 테이블에서도 삭제된 상태
- 영속 상태의 엔티티는 EntityManager.remove 메소드를 이용해서 제거할 수 있음

영속 컨텍스트와 엔티티 상태

2 영속 컨텍스트와 1차 캐시

〈1〉 1차 캐시란?

- persist() 메소드로 영속성 컨텍스트에 등록한 엔티티는 영속 컨텍스트에 존재하는 1차 캐시에 등록됨
- 1차 캐시는 Key-Value 형태로 엔티티를 저장하고 관리

2 영속 컨텍스트와 1차 캐시

〈1〉 1차 캐시란?

1차 캐시에 저장된 엔티티의 Key

- 식별자 값 저장

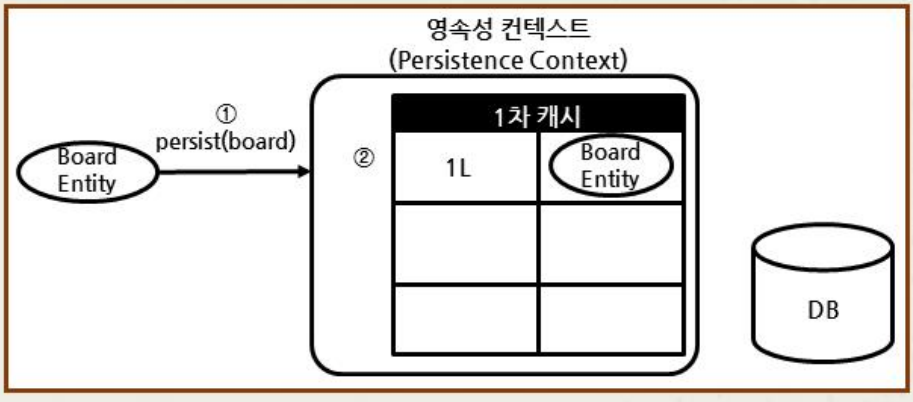
1차 캐시에 저장된 엔티티의 Value

- 실제 엔티티 저장

영속 컨텍스트와 엔티티 상태

2 영속 컨텍스트와 1차 캐시

〈1〉 1차 캐시란?



2 영속 컨텍스트와 1차 캐시

〈2〉 1차 캐시와 트랜잭션

- 1차 캐시에 등록된 엔티티는 트랜잭션이 종료될 때 데이터베이스에 반영됨

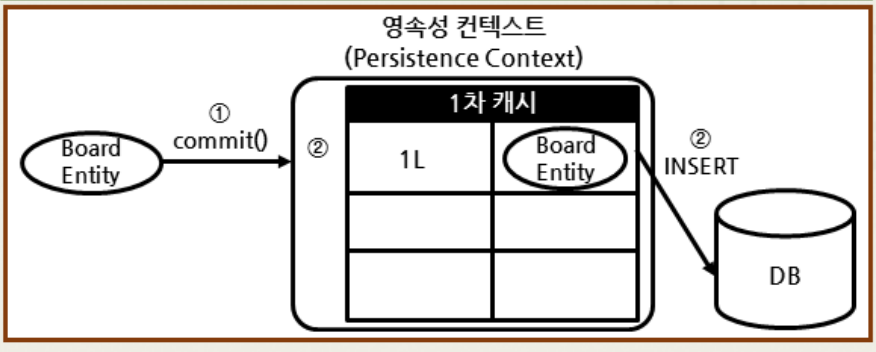
영속 컨텍스트와 엔티티 상태

2 영속 컨텍스트와 1차 캐시

〈2〉 1차 캐시와 트랜잭션

플러시(Flush)

- 영속 컨텍스트에 저장된 엔티티를 데이터베이스에 반영하는 과정



영속 컨텍스트와 SQL 저장소

1 영속 컨텍스트와 SQL 저장소

〈1〉 SQL 저장소란?

- 영속 컨텍스트는 1차 캐시 뿐만 아니라 SQL 저장소도 가지고 있음

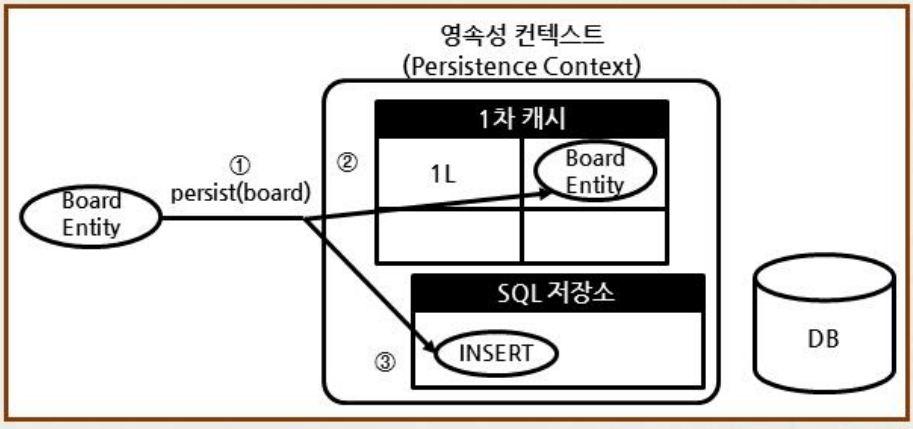
1 영속 컨텍스트와 SQL 저장소

〈1〉 SQL 저장소란?

영속 컨텍스트와 SQL 저장소

1 영속 컨텍스트와 SQL 저장소

〈1〉 SQL 저장소란?



1 영속 컨텍스트와 SQL 저장소

〈2〉 플러시(Flush)

- 새로운 엔티티가 등록될 때마다 1차 캐시와 SQL 저장소에는 각각 엔티티와 SQL 구문들이 누적됨

영속 컨텍스트와 SQL 저장소

1 영속 컨텍스트와 SQL 저장소

〈2〉 플러시(Flush)

1. EntityManager.commit 메소드로 트랜잭션을 종료
2. SQL 저장소에 저장된 모든 SQL이 한번에 데이터 베이스에 전송됨

1 영속 컨텍스트와 SQL 저장소

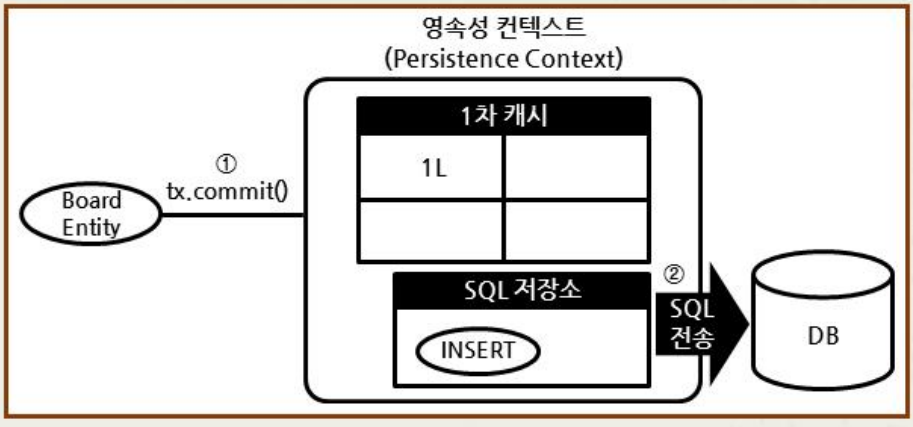
〈2〉 플러시(Flush)

- 한 번의 데이터베이스 연결로 SQL 구문들을 처리하기 때문에 성능을 최적화할 수 있음

영속 컨텍스트와 SQL 저장소

1 영속 컨텍스트와 SQL 저장소

〈2〉 플러시(Flush)



2 엔티티 수정과 더티 체크

〈1〉 JPA의 UPDATE

- EntityManager에는 엔티티를 수정하기 위한 메소드를 별도로 제공하지 않음

영속 컨텍스트와 SQL 저장소

2 엔티티 수정과 더티 체크

〈1〉 JPA의 UPDATE

2 엔티티 수정과 더티 체크

〈1〉 JPA의 UPDATE

- JPA의 UPDATE는 기본적으로 모든 칼럼 수정을 원칙으로 함
- 모든 경우의 UPDATE를 하나의 SQL로 처리할 수 있기 때문

영속 컨텍스트와 SQL 저장소

2 엔티티 수정과 더티 체크

〈1〉 JPA의 UPDATE

- 실제 변경된 변수만 UPDATE에 포함시키고 싶은 경우

• 엔티티 클래스에 @DynamicUpdate를 적용

2 엔티티 수정과 더티 체크

〈2〉 스냅샷(Snapshot)이란?

• 어떤 상태를 기록하기 위해서 사진을 찍듯
특정 객체의 상태를 저장하는 것

• 영속 컨텍스트는 엔티티가 1차 캐시에 등록되는 순간
복사본의 객체를 스냅샷을 찍어 저장함

영속 컨텍스트와 SQL 저장소

2 엔티티 수정과 더티 체크

〈3〉 더티 체크(Dirty Check)란?

1. 엔티티를 수정하고 트랜잭션을 종료
2. 영속 컨테이너는 1차 캐시에 등록된 엔티티 중에서 스냅샷과 상태가 다른 엔티티들을 검색함

2 엔티티 수정과 더티 체크

〈3〉 더티 체크(Dirty Check)란?

- 영속 컨테이너가 엔티티 캐시에 등록된 엔티티 중에서 값이 변경된 엔티티들을 찾는 과정
- Dirty는 더럽다는 의미가 아닌 **변경을 의미함**

적용 스프링 부트


```java
@Service
public class BoardServiceRunner implements ApplicationRunner {

    @Override
    public void run(ApplicationArguments args) throws Exception {
        // EntityManagerFactory 생성
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("BoardWeb");

        // EntityManager 생성
        EntityManager em = emf.createEntityManager();

        // EntityTransaction 생성
        EntityTransaction tx = em.getTransaction();

        // Transaction 시작
        tx.begin();

        // 데이터 등록
        Board board = new Board();
        board.setTitle("MyBatis 제목");
        board.setWriter("채규태");
        board.setContent("MyBatis 내용.............");
        em.persist(board);

        // Transaction 종료
        tx.commit();

        // EntityManagerFactory, EntityManager 종료
        em.close();
        emf.close();


    }
```

실습단계
JPA를 이용해 CRUD 기능 구현
영속성 컨텍스트, 1차 캐시, SQL 저장소, 더티 체크
보드 엔티티 한 개를 생성해 영속 컨텍스트에 등록하면 내부적으로 insert 발생
여러 데이터를 등록한다고 가정
만약 데이터 입력 건수가 늘어날 때 매번 Connection을 연결하는 것은 퍼포먼스 상에 문제가 있음
여러 insert나 delete를 하나의 트랜젝션 범위 안에서 처리하는 것이 SQL 저장소의 JPA 내부적 운영 목적
비영속 상태
보드 엔티티를 영속 컨텍스트에 등록
1차 캐쉬에 등록됨
SEQ 변수에 저장된 값이 키 값으로 사용됨
엔티티 객체들을 영속 컨텍스트 안에서 식별할 수 있음
엔티티 객체에 설정된 값들을 기반으로 INSERT SQL을 SQL 저장소에 generation 해서 등록

적용 스프링 부트

실습단계
트랜젝션을 종료하려고 commit method를 호출하는 순간
한 번의 커넥션을 통해 DB로 전달됨
한 번의 커넥션으로 관련 DML 작업을 한꺼번에 처리할 수 있음
중요! 한 번의 커넥션을 통해 10건 데이터 등록 가능
수정 : 상세조회한 엔티티의 제목과 내용 수정
내부적으로 업데이트 동작
업데이트 관련한 메소드는 존재하지 않음
내부적으로 더티 체크 동작
더티 체크 : 원래 상태에서 변형된 엔티티가 있는지 체크
더티 체크 : 변경된, 수정된 엔티티가 있는지 체크해서 Update SQL을 등록하는 과정
Insert SQL이 SQL 저장소에 등록
1번 엔티티에 대한 수정 작업 발생
Update SQL이 제너레이션 되어 수정된 것을 확인 가능
삭제 기능 구현
업데이트는 더티 체크를 이용해 처리
Delete SQL 제너레이션됨
1번 엔티티가 셀렉트 목록에서 사라짐
JPQL : 목록 조회 시 JPA가 제공하는 별도 쿼리 명령어 사용
검색 기준이 테이블이 아닌 엔티티
컬럼 이름이 아닌 Board 엔티티의 seq 변수
JPQL을 실행하려면 createQuery 메소드를 호출해야 함
실행 10건의 데이터 중에서 5건의 데이터만 검색됨