JPA 부숴

개요

- 어려운 이유
- - 객체와 테이블 올바른 매핑, 설계하는 방법 모름
  - JPA 내부 동작 방식 이해를 못함(디버깅 오래걸리고, 장애 상황에 대처못함)
- 장점
- - 단순 SQL 작성으로 인생을 낭비하지 않을 수 있대

JPA 소개

- 객체를 관계형 DB에 관리해야함
- 관계형DB 관리할라면 SQL에 의존 해야 한다는게 문제
  - 반복적인 작업, 필드 변경시 다 바꿔줘야함
- 패러다임의 불일치(객체 vs 관계형DB)
- 객체와 관계형DB의 차이
  - 상속
  - 연관관계(reference, pk-fk)
    - 객체는 member.getTeam() 이런식으로, 테이블은 외래 키 사용
    - 객체는 단방향인데 테이블은 양방향이네?!
- 객체답게 모델링 할수록 매핑 작업만 늘어남
- 객체를 자바 컬렉션에 저장 하듯이 관계형DB에 저장할 수는 없을까? => 그에 대한 답이 JPA

JPA란?

- 자바 진영의 ORM 기술 표준
  - Object Relational Mapping
  - 객체는 객체대로 설계, 관계형DB는 관계형DB대로 설계
  - ORM 프레임워크가 중간에서 매핑
- java app과 jdbc API 사이에서 동작
- 왜 사용해야는지?
  - SQL 중심 개발에서 객체 중심으로 개발
  - 생산성
    - 저장:persist(), 조회 find(), 수정 setName(), 삭제 remove()
  - 유지보수
    - 필드 변경시 모든 SQL 수정하던거 JPA는 필드만 수정하면 됨
  - 패러다임의 불일치 해결

        - 상속, 연관관계 이런거
    - 성능

- JPA의 성능 최적화 기능

    - 1차 캐시와 동일성 보장(같은 트랜잭션 안에서는 같은 엔티티 반환)
    - 트랜잭션을 지원하는 쓰기 지연(JDBC batch 기능인데 쓸라면 복잡, jpa에선 기능 하나만 켜주면 됨)
    - 지연 로딩과 즉시 로딩(jpa에서 옵션 키면 됨)
        - 지연 로딩: 객체가 실제 사용될 때 로딩
        - 즉시 로딩: JOIN SQL로 한번에 연관된 객체까지 미리 조회

- JPA 구동 방식

    - Persistence => 생성 => EntityManagerFactory => 생성 => EntityManager...

```
public class JpaMain {
    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("hello");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();
        tx.begin();

        try {
            Member member = new Member(100L, "hello");
            member.setName("HelloJPA");
            tx.commit();
        } catch (Exception e){
            tx.rollback();
        } finally {
            em.close();
            emf.close();
        }
    }
}
```

    - JPA는 모든 단위 트랜잭션 안에서 작동해야함
    - setName() 이런식으로 해도 변경됨 부를때 jpa에서 알아서 관리함

- 주의

    - EntityManagerFactory는 하나만 생성해서 애플리케이션 전체에 공유
    - EntityManager는 스레드간에 공유x, 사용하고 버려야함
    - JPA의 모든 데이터 변경은 트랜잭션 안에서 실행

- JPQL

    - 대상이 테이블이 아니고 객체가 대상이 됨

    - 객체에 맞춘 쿼리로 이거로 짜면 각각 db에 맞게 변환해줌

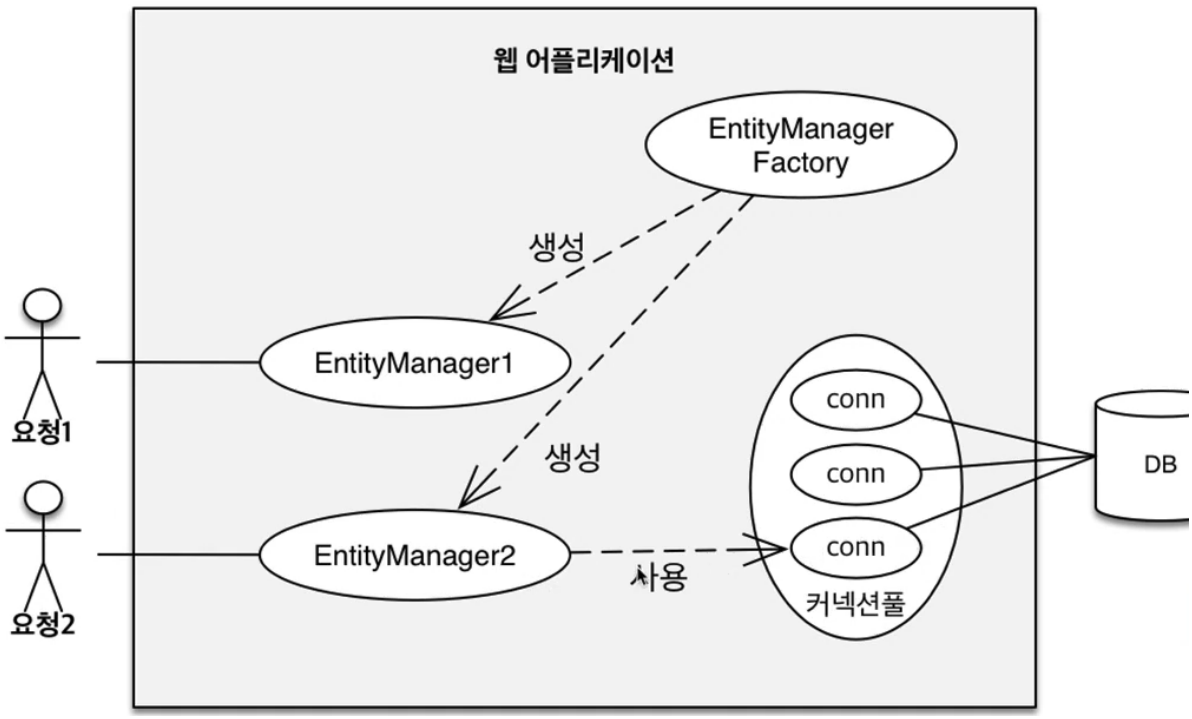
- 문제는 검색 쿼리
- app에서 필요한 데이터만 DB에서 불러오려면 결국 검색 조건이 포함된 SQL이 필요
- 따라서 JPA는 SQL을 추상화한 JPQL이라는 객체 지향 쿼리 언어 제공

JPA 가장 중요한 두가지

- 객체와 관계형DB 매핑하기
- 영속성 컨텍스트

영속성 컨텍스트

엔티티 매니저 팩토리와 엔티티 매니저



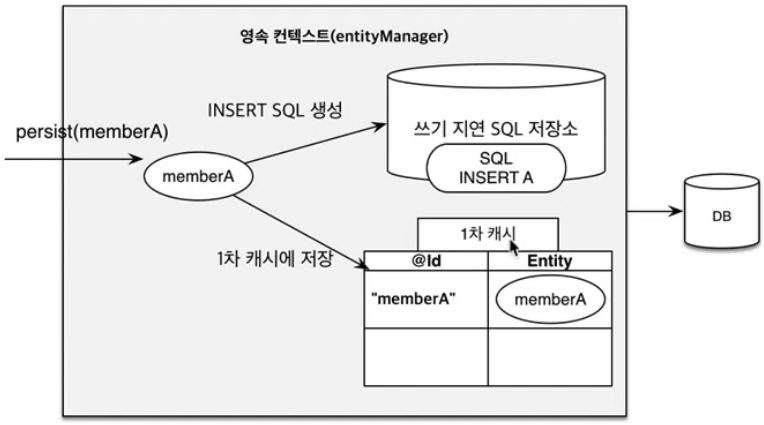
- JPA를 이해하는데 가장 중요한 요어로, entity를 영구 저장하는 환경
- 논리적인 개념으로, EntityManager를 통해서 영속성 컨텍스트에 접근
- Entity의 생명주기
  - 비영속(new/transient): 영속성 컨텍스트와 전혀 관계가 없는 새로운 상태
  - 영속(managed): 영속성 컨텍스트에 관리되는 상태
  - 준영속(detached): 영속성 컨텍스트에 저장되었다가 분리된 상태
  - 삭제(removed): 삭제된 상태

- 커밋될때 영속석 컨텍스트 같이 넘어감
- 영속성 컨텍스트의 이점
  - 1차 캐시
    - pk가 key이고, 객체 자체가 값으로 저장됨
    - find()하면 1차 캐시에서 먼저 조회함
    - EntityManager가 트랜잭션 단위라 사실 그렇게 속도에 이점은 안됨
  - 동일성(identity) 보장
    - 1차 캐시로 반복 가능한 읽기(REPEATABLE READ)등급의 트랜잭션 격리 수준을 DB가 아닌 app 차원에서 제공

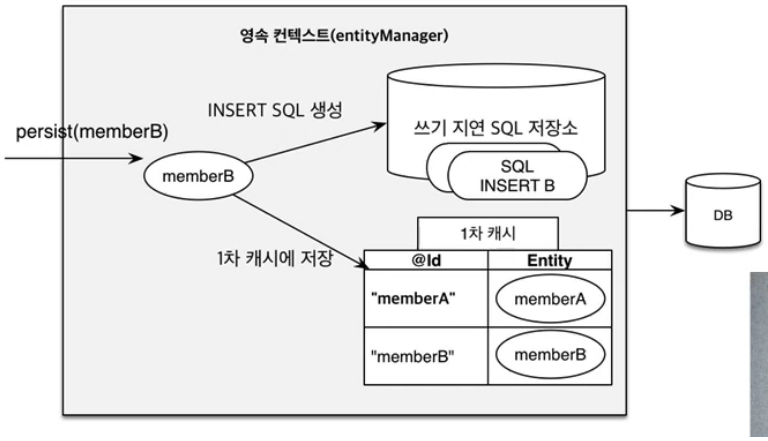
```
Member a = em.find(Member.class, "member1");
Member b = em.find(Member.class, "member1");
System.out.println(a == b);     // true
```

  - 트랜잭션을 지원하는 쓰기 지연(transactional write-behind)

em.persist(memberA);

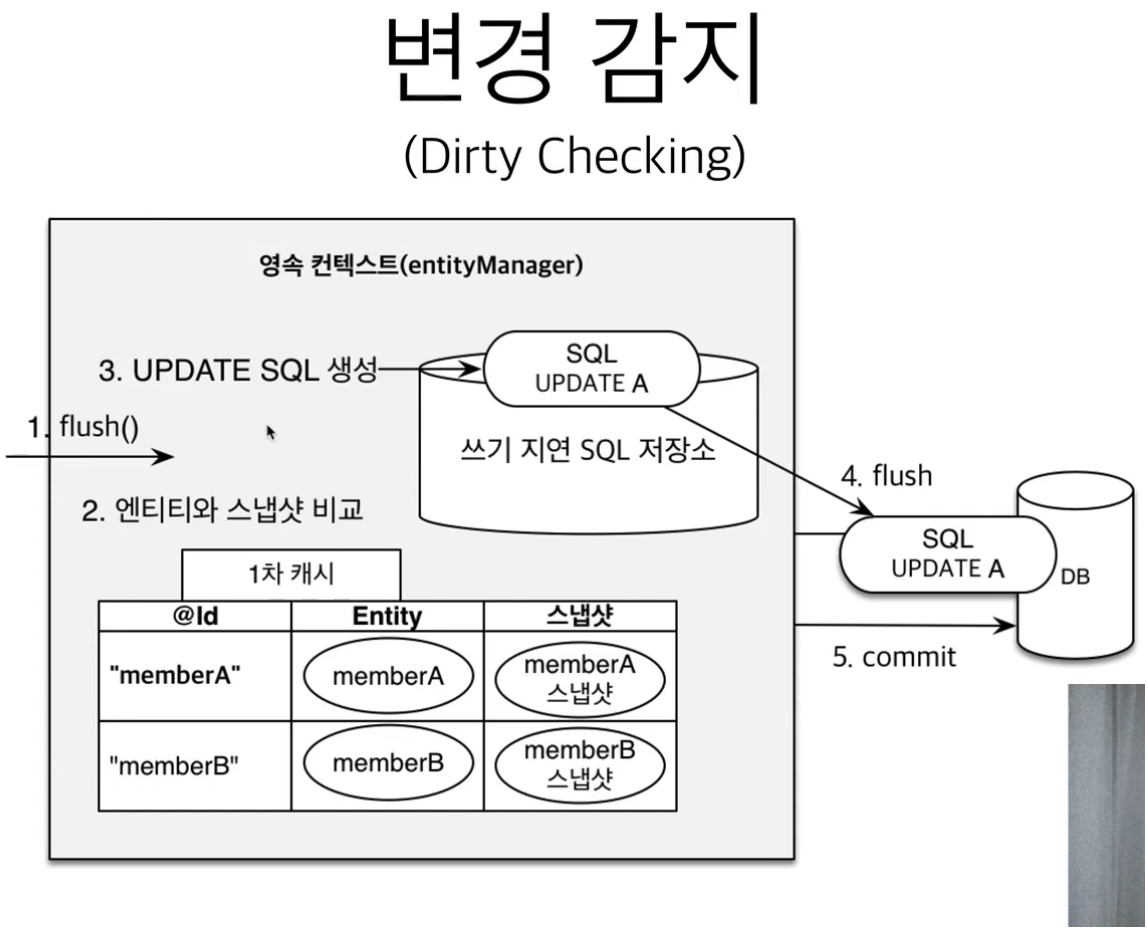


em.persist(memberB);



    - 커밋하는 시점에 flush됨
  - 변경 감지(Dirty Checking)

```
Member memberA = em.find(Member.class, "memberA");
member.setAge(10);
//em.update(member); 이런 코드가 없어도 DB 값 변경 됨!
```



    - 커밋하면 entity와 스냅샷(값 읽어온 최초시점 상태) 비교
    - 바뀐 값이 있으면 update query 저장소에 추가

  - 지연 로딩(Lazy Loading)

- 플러시
  - 영속성 컨텍스트의 변경내용을 DB에 반영
  - 발생하면 dirty checking해서 수정된 entity 지연 SQL 저장소에 등록하고, 저장소 쿼리들 DB에 전송
  - 1차 캐시 지우거나 하는건 아니고 바뀐거 저장소에 추가하고 저장소 쿼리 DB에 반영되는 과정
  - 영속성 컨텍스트를 비우진 않고 변경 내용을 DB에 동기화, 트랜잭션 이라는 작업단위가 중요 => 커밋 직전에만 동기화 하면 됨
  - 플러시 하는 방법
    - em.flush(), 직접 호출(보통 이렇게 쓸일은 없는데 테스트 할 때 필요할수도)
    - commit()
    - JPQL 쿼리 실행
  - 옵션
    - FlushModeType.AUTO: 기본값으로, 커밋이나 쿼리 실행시 플러시

- 준영속(detached) 상태
    - cf) 영속상태 => persist()나 find()해서 1차 캐시에 올라가 있는 상태
    - detach(), clear(), close() 등이 있음