JPA 3장

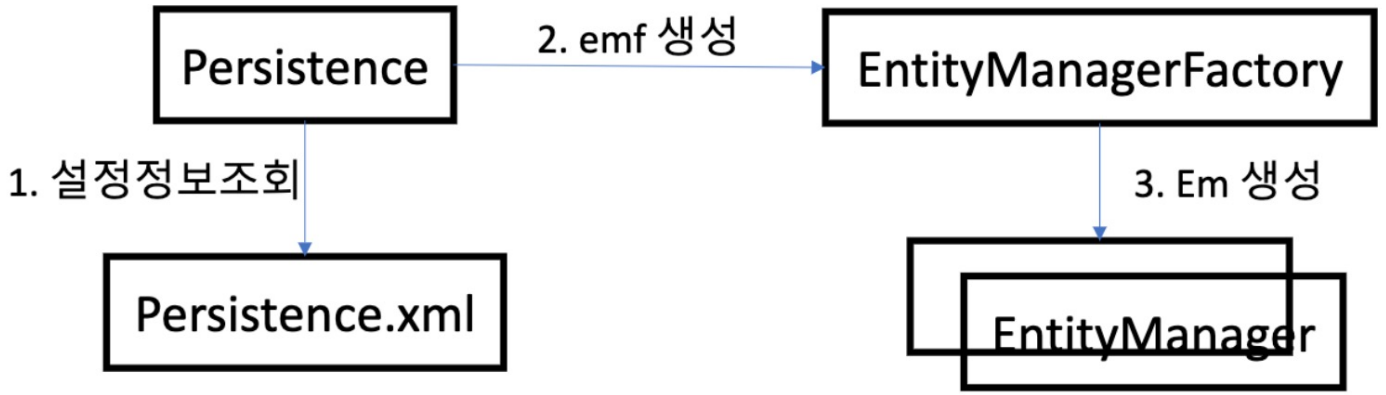
참조: 자바 ORM 표준 JPA 프로그래밍(김영한)

3장 - 영속성 관리

1) 엔티티 매니저 팩토리와 엔티티 매니저
2) 영속성 컨텍스트란?
3) 엔티티의 생명주기
4) 영속성 컨텍스트의 특징
5) 플러시
6) 준영속
7) 정리

복습

엔티티 매니저 설정



EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("jpastudy");

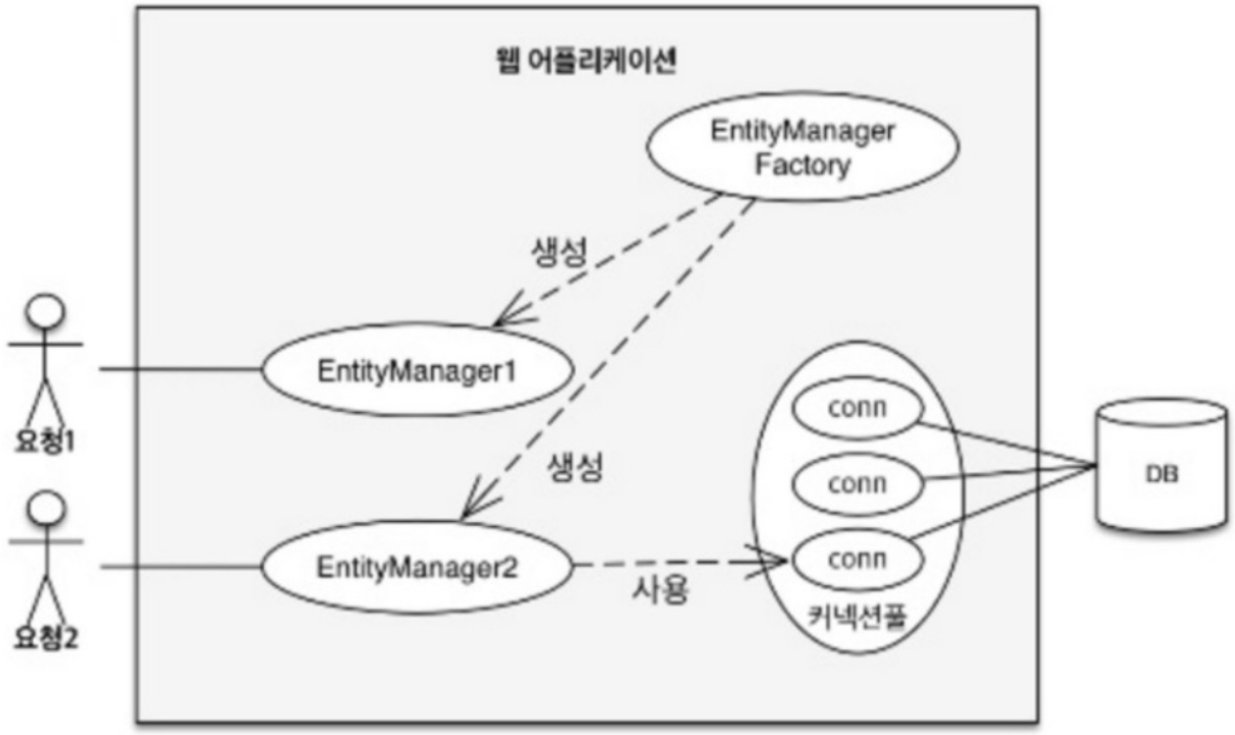
- JPA 시작을 위해 persistence.xml의 설정 정보를 사용해 emf생성
- <persistence-unit name="jpastudy">을 찾아 emf생성하며 emf는 전체에서 딱 한 번만 생성

EntityManager em = emf.createEntityManager();

- em을 통해 DB에 엔티티를 등록.수정.삭제.조회 가능(CRUD)
- 개발자는 em을 가상의 DB로 생각할 수 있다
- 공유, 재사용 X

복습

em 공유가 안되는 이유



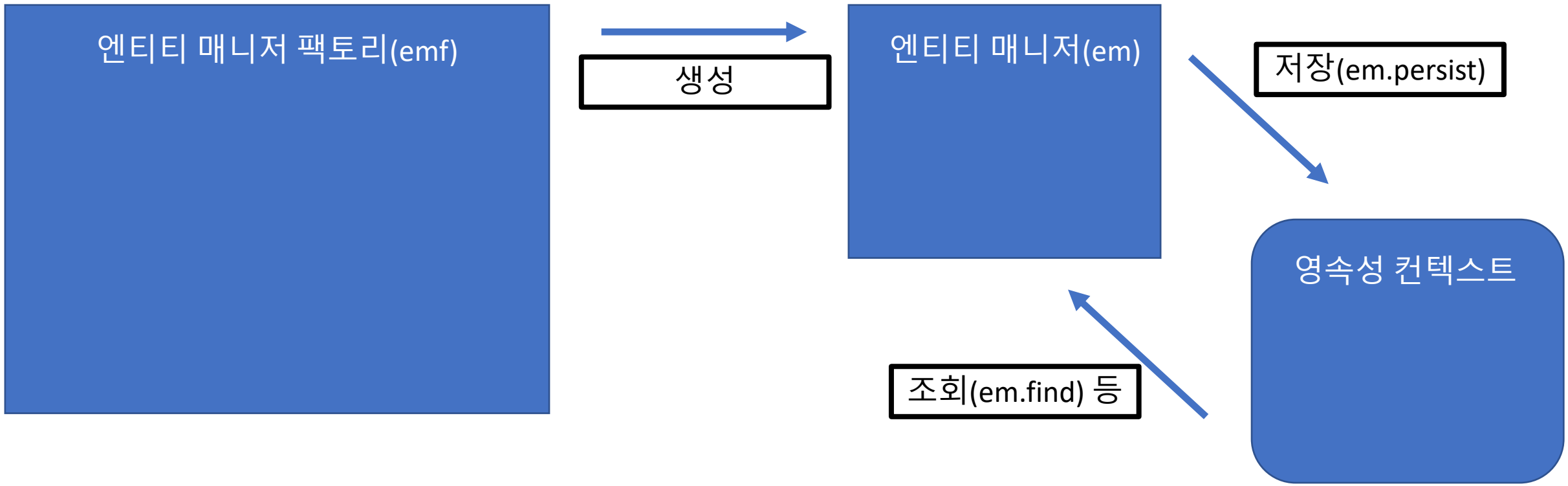
출처: https://debaeloper.tistory.com/38

em은 **데이터베이스 커넥션과 밀접하게** 관계가 있다.
따라서 여러 쓰레드가 동시 접근하면 **동시성 문제**가 발생하기에 **쓰레드간 공유를 하면 안된다.**

동시성 문제: 여러 스레드가 동시에 같은 인스턴스의 필드 값을 변경 하면서 발생하는 문제를 의미

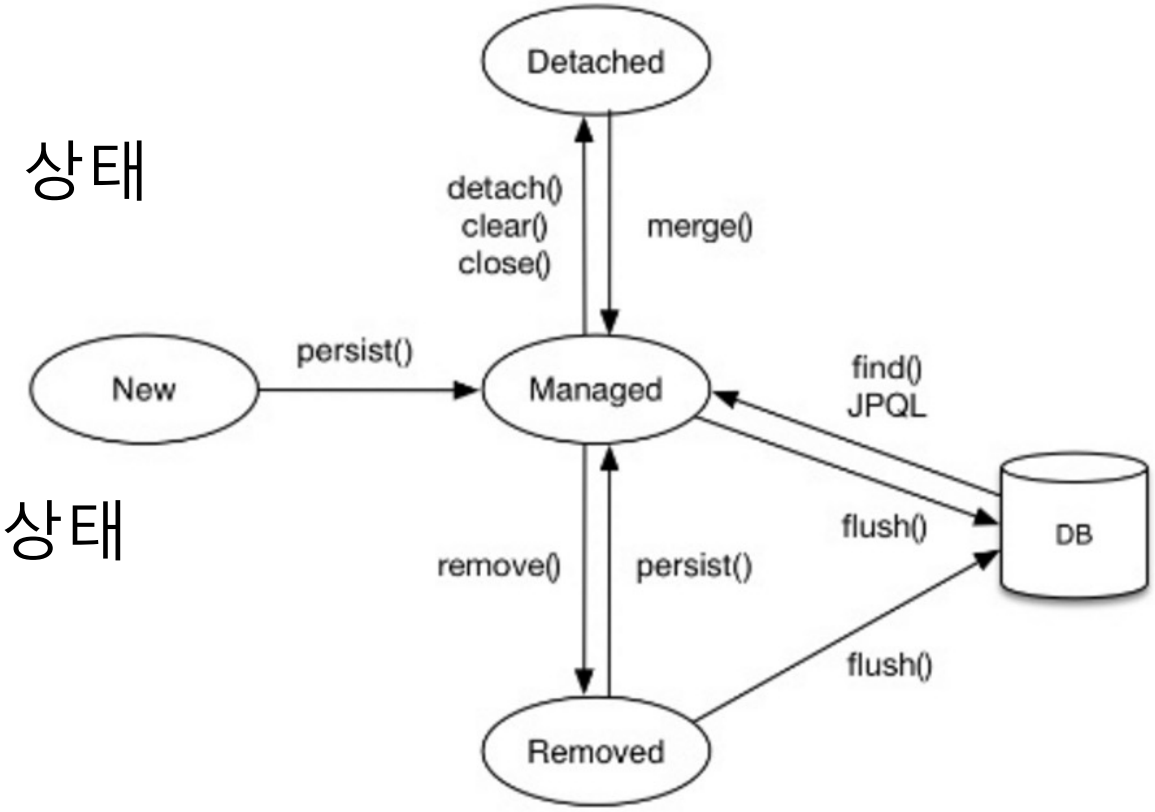
2) 영속성 컨텍스트(persistence context)란?

- JPA를 이해하는 데 가장 중요한 용어
- 엔티티 매니저를 사용해 영속성 컨텍스트를 관리할 수 있음



3) 엔티티의 생명주기

- 비영속(new/transient) : 전혀 관계가 없는 상태
- 영속(managed) : 저장된 상태
- 준영속(detached) : 저장되었다가 분리된 상태
- 삭제(removed) : 삭제된 상태

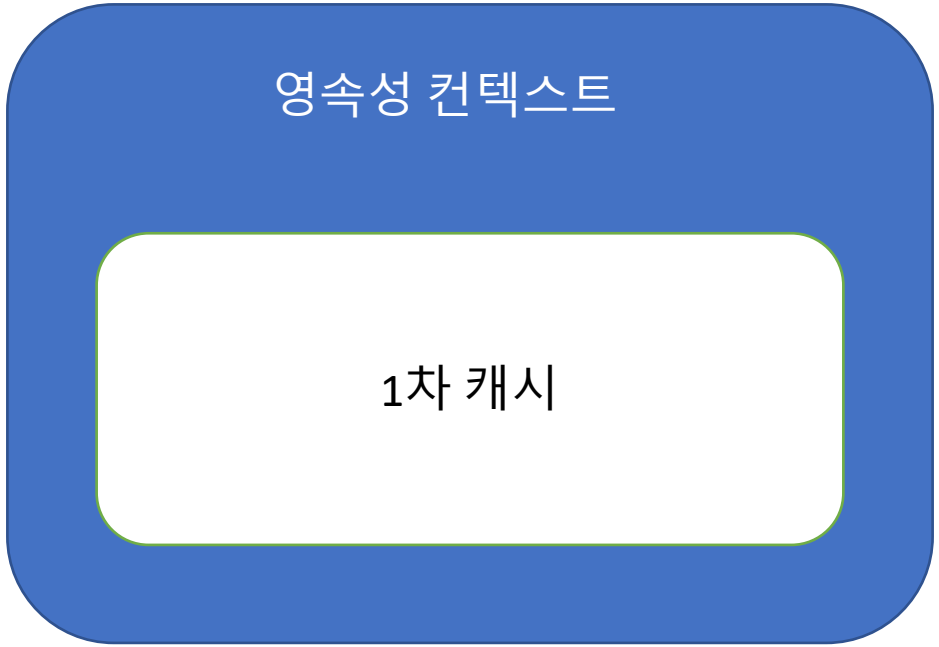


4) 영속성 컨텍스트의 특징

- 식별자 값
  - 영속 상태는 식별자 값이 반드시 있어야 함 (`@Id`로 매핑)
- 데이터베이스 저장
  - 영속성 컨텍스트에 저장된 엔티티들을 데이터베이스에 반영하는 것

    -> 플러시(flush)라고 함
- 여러 장점들
  - 1차 캐시, 동일성 보장, 트랜잭션을 지원하는 쓰기 지연, 변경 감지, 지연 로딩

엔티티 조회



영속 상태의 엔티티는 모두 **1차 캐시**에 저장됨

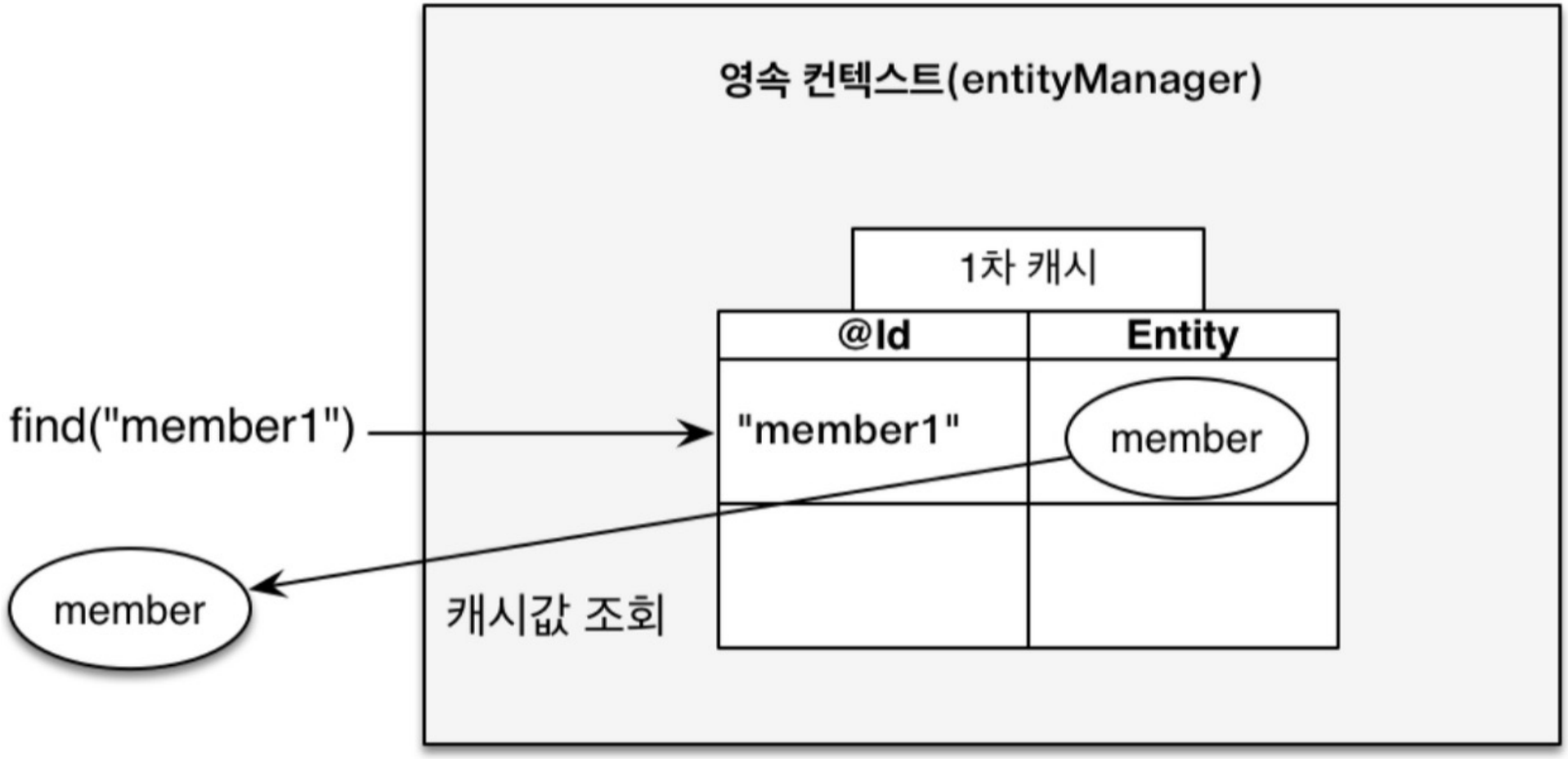
엔티티 조회 코드

```
Member member =
                em.find(Member.class, “member1”);
```

```
em.find(Member.class, “member1”);
```

위 메소드로 엔티티를 조회하면 1차 캐시, 데이터베이스 순서로
값을 조회하게 된다

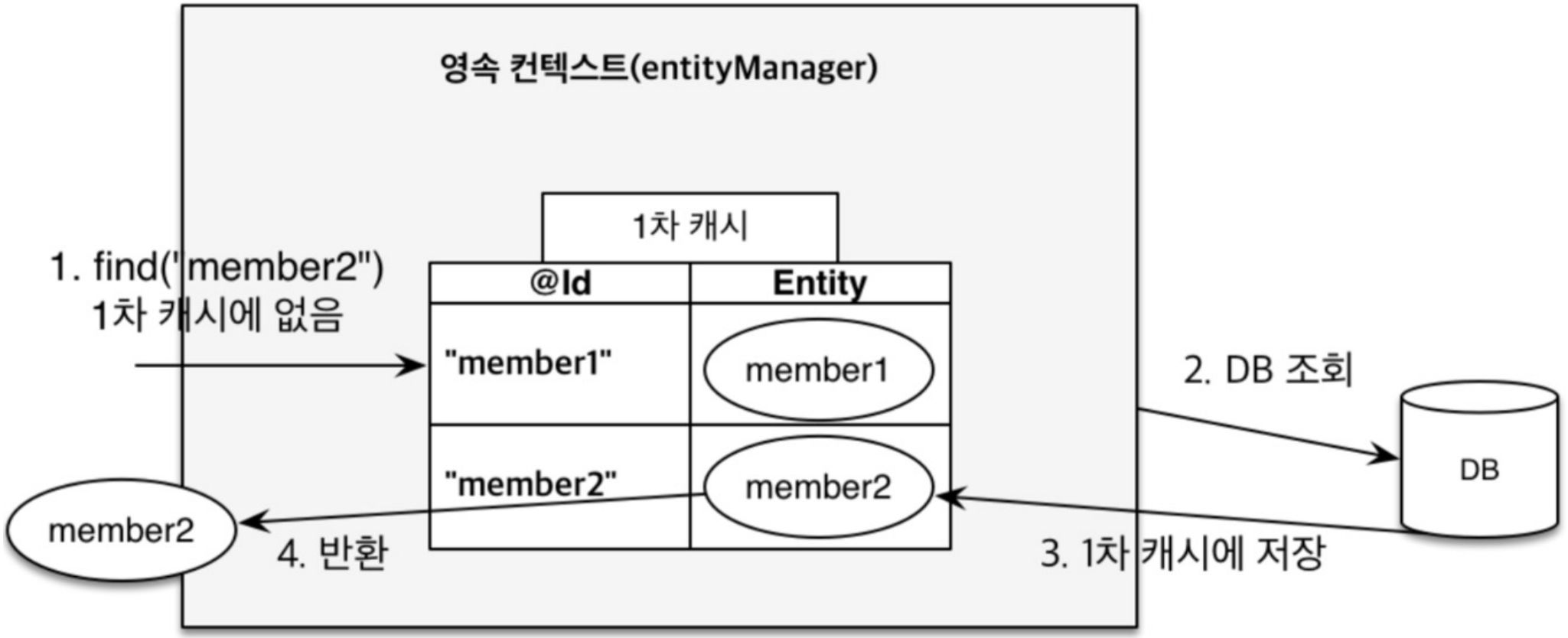
엔티티 조회(이어서)



<1차 캐시에서 조회하는 경우>

데이터베이스 조회 x

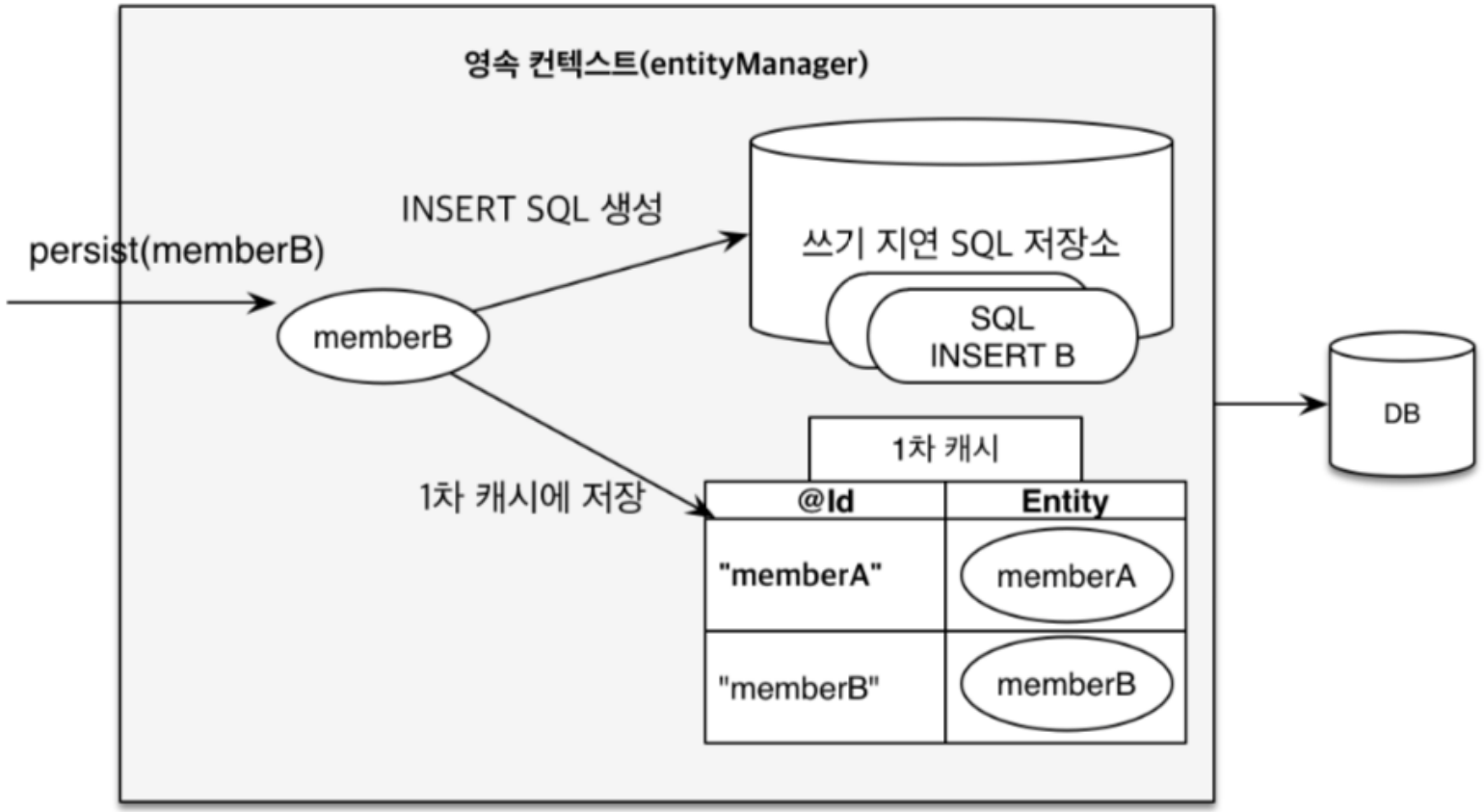
엔티티 조회(이어서)



<데이터베이스에서 조회하는 경우>

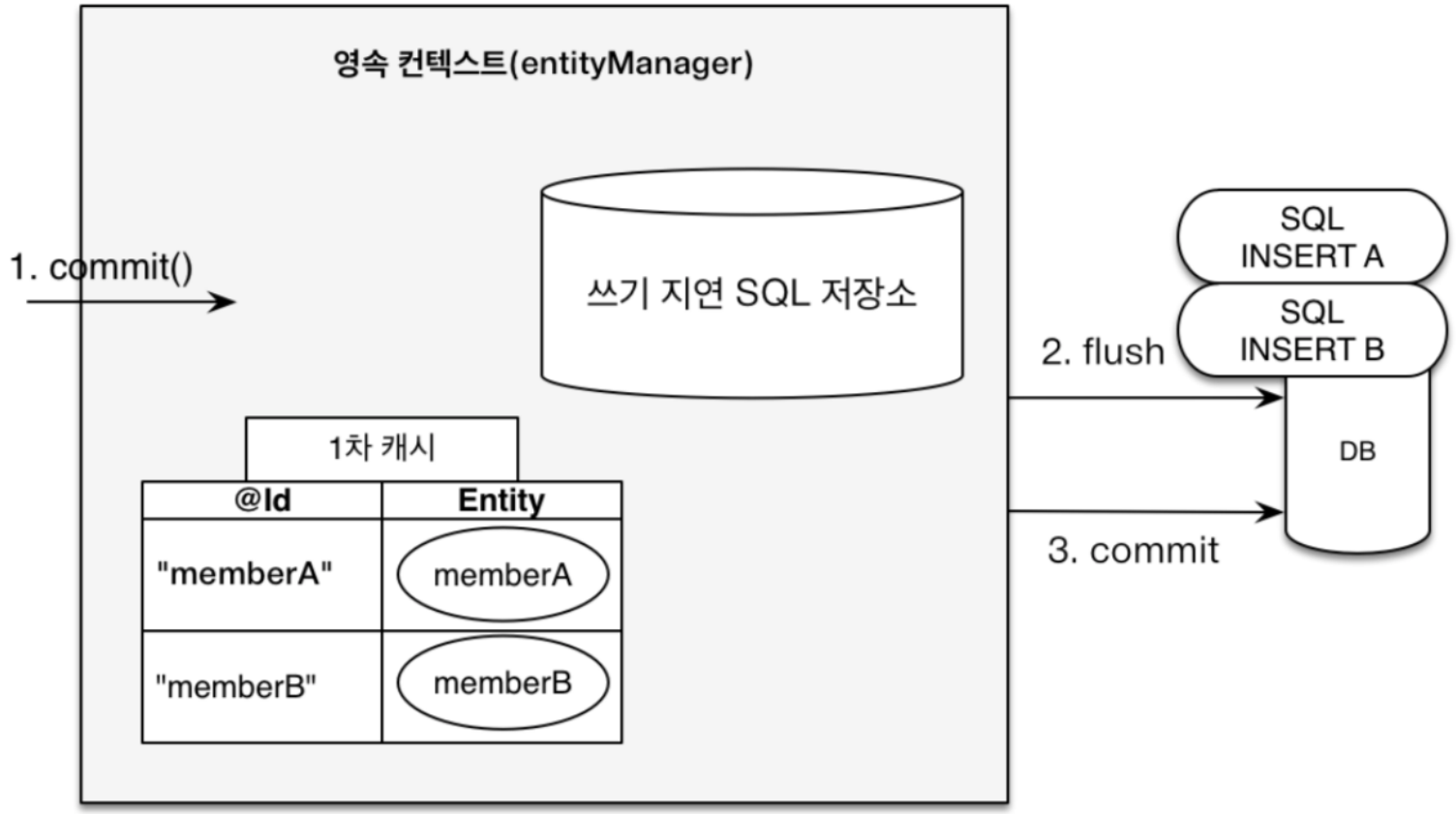
1차 캐시에 해당 엔티티를 저장 후 반환 함

엔티티 등록



엔티티 매니저는 트랜잭션을 커밋하기 직전까지 DB에 직접 저장 X
그 대신 INSERT SQL을 차곡차곡 쌓아둠

엔티티 등록(이어서)



트랜잭션을 커밋할 때 모아둔 쿼리를 DB에 보냄 -> **트랜잭션을 지원하는 쓰기 지연**

엔티티 수정

엔티티를 조회해서 값을 변경한 후

`em.update()`같은 메소드를 사용해서 수정하면 될까?

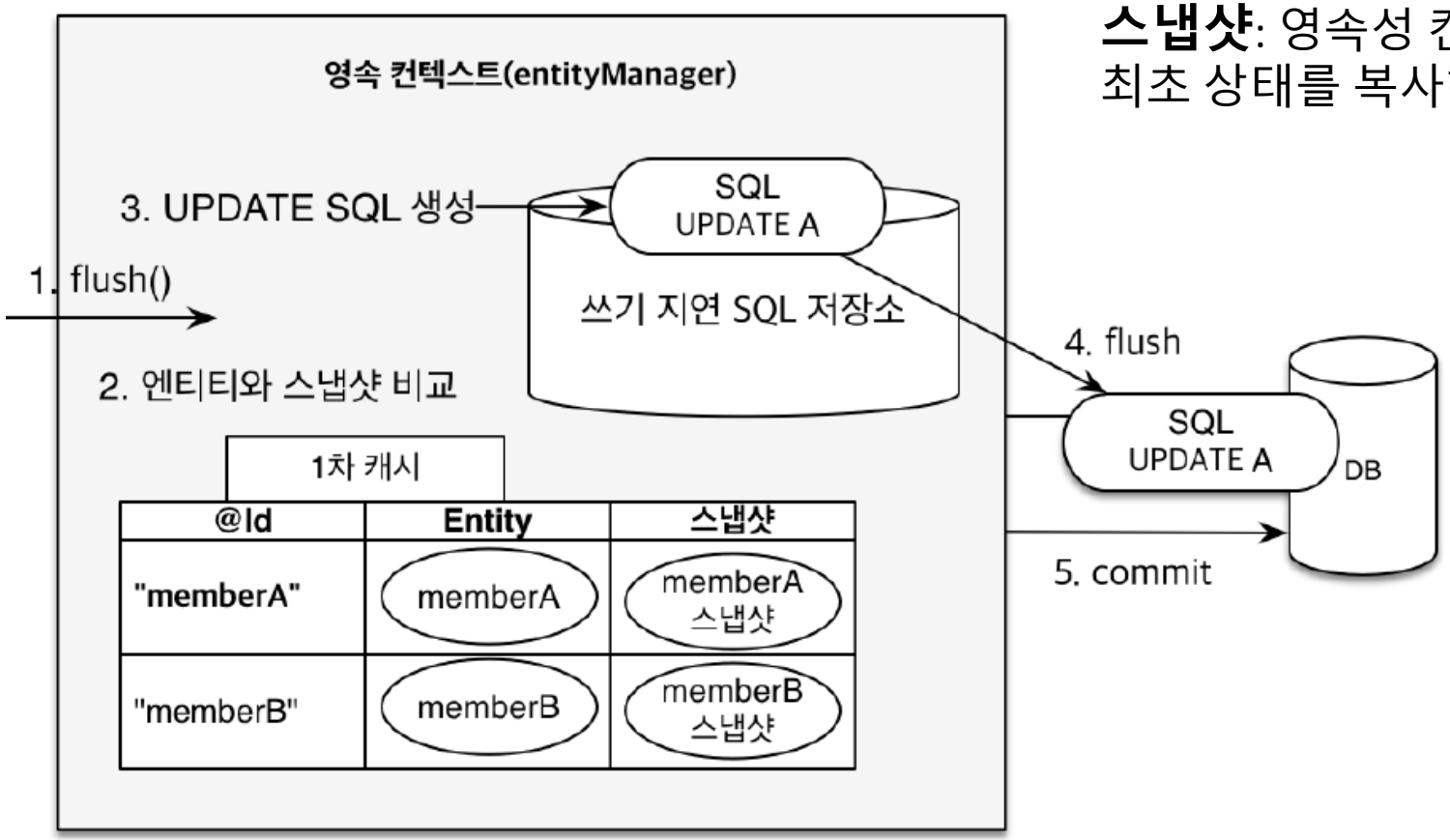
-> 엔티티 매니저에는 `em.update()` 같은 메소드는 없음

결론부터 말하자면,

단순히 엔티티를 조회해서 데이터를 변경하기만 하면 자동으로 수정이 된다.

어떻게?? -> **변경 감지 (dirty checking)**

엔티티 수정(이어서)



스냅샷: 영속성 컨텍스트에 엔티티를 보관할 때,
최초 상태를 복사해서 저장해두는 것

1. 트랜잭션 커밋시 플러시가 호출 됨
2. 엔티티와 스냅샷 비교해서 변경된 엔티티 찾기
3. 변경된 엔티티가 있을 경우 수정 쿼리를 생성
4. 해당 쿼리를 DB에 보냄
5. DB 트랜잭션을 커밋

엔티티 삭제

엔티티를 삭제하려면 먼저 해당 엔티티를 조회해야 함

```
Member memberA = em.find(Member.class, “memberA”);
//삭제 대상 엔티티 조회
em.remove(memberA);
//엔티티 삭제
```

5) 플러시 (flush( ))

영속성 컨텍스트의 변경 내용을 DB에 반영하는 것 (동기화)

1. 직접 호출
    - em.flush( ) 메소드를 직접 호출하는 방법
2. 트랜잭션 커밋 시 플러시 자동 호출
    - DB에 트랜잭션 커밋 전 꼭 플러시를 호출해서 영속성 컨텍스트의 변경 내용을 반영해줘야 함
    - 이를 위해 JPA 에서는 트랜잭션 커밋 시 플러시를 자동으로 호출해줌
3. JPQL 쿼리 실행 시 플러시 자동 호출
    - JPQL이나 Criteria 같은 객체지향 쿼리를 호출해도 플러시가 실행 됨

3. JPQL 쿼리 실행 시 플러시 자동 호출

```
em.persist(memberA);
em.persist(memberB);
em.persist(memberC);

//중간에 JPQL 실행
query = em.createQuery("select m from Member m", member.class);
List<Member> members = query.getResultList();
```

영속성 컨텍스트에만 있는 엔티티를 DB에서 조회하게 되면 결과가 조회되지 않음

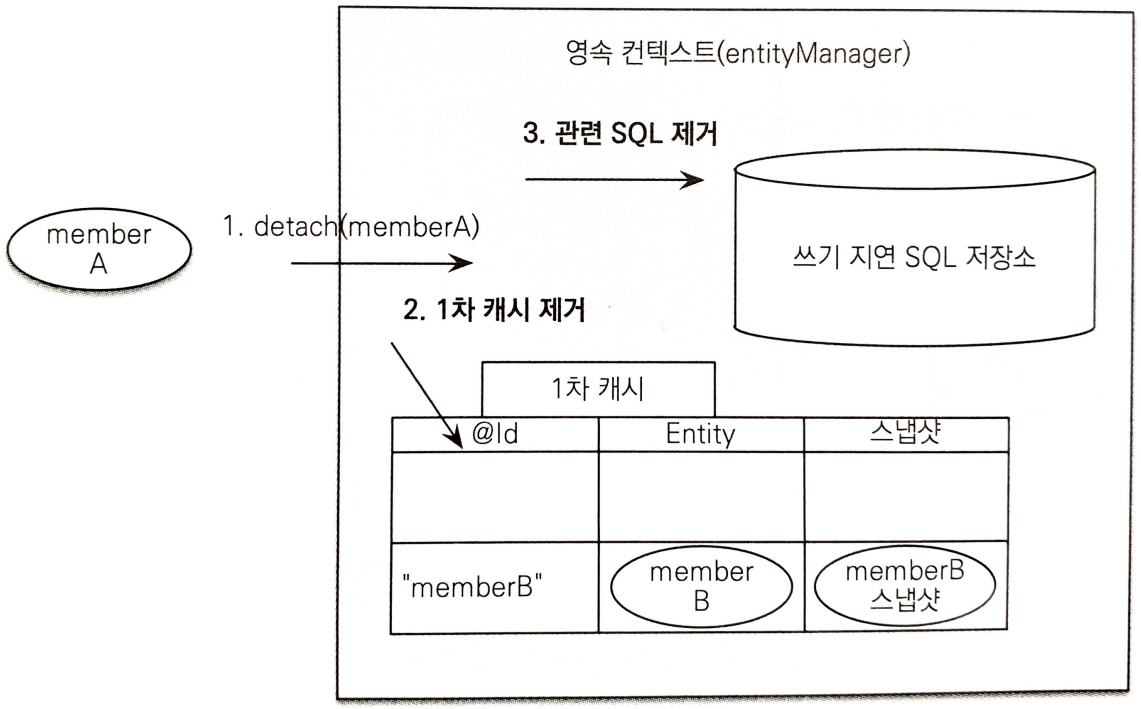
-> 따라서 쿼리 실행 직전에 영속성 컨텍스트를 플러시 해서 변경 내용을 DB에 반영해야 함

6) 준영속

영속 상태의 엔티티를 준영속 상태로 만드는 방법 3가지

1. `em.detach(entity)`
    - 특정 엔티티만 준영속 상태로 전환한다.
2. `em.clear()`
    - 영속성 컨텍스트를 완전히 초기화한다.
3. `em.close()`
    - 영속성 컨텍스트를 종료한다.

6) 준영속


영속 컨텍스트(entityManager)
3. 관련 SQL 제거
member
A
1. detach(memberA)
쓰기 지연 SQL 저장소
2. 1차 캐시 제거
1차 캐시
@Id
Entity
스냅샷
"memberB"
member
B
memberB
스냅샷

참고 사이트

- https://leejaedoo.github.io/persistence/