

# 영속성 관리

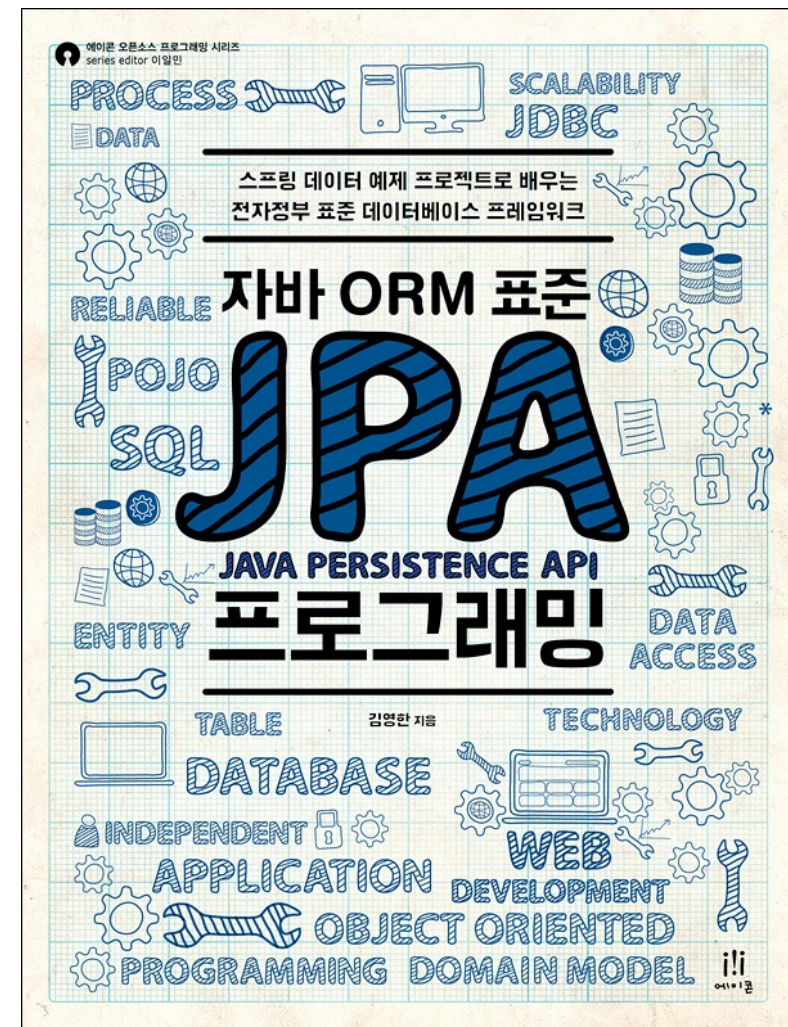
---

JPA 내부 구조

# 김영한

SI, J2EE 강사, DAUM, SK 플래닛,  
우아한형제들

저서: 자바 ORM 표준 **JPA** 프로그래밍

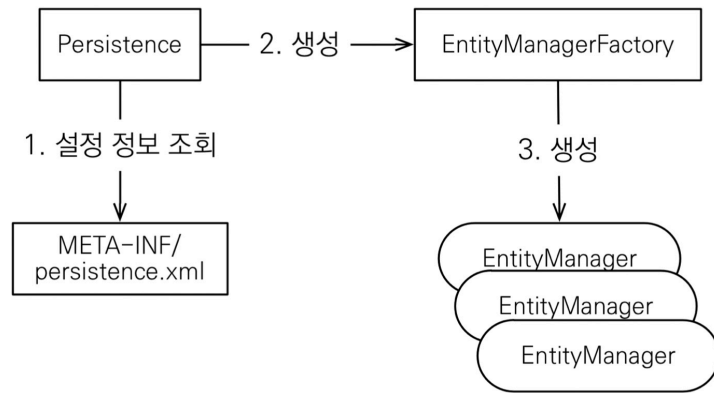


영속성 컨텍스트

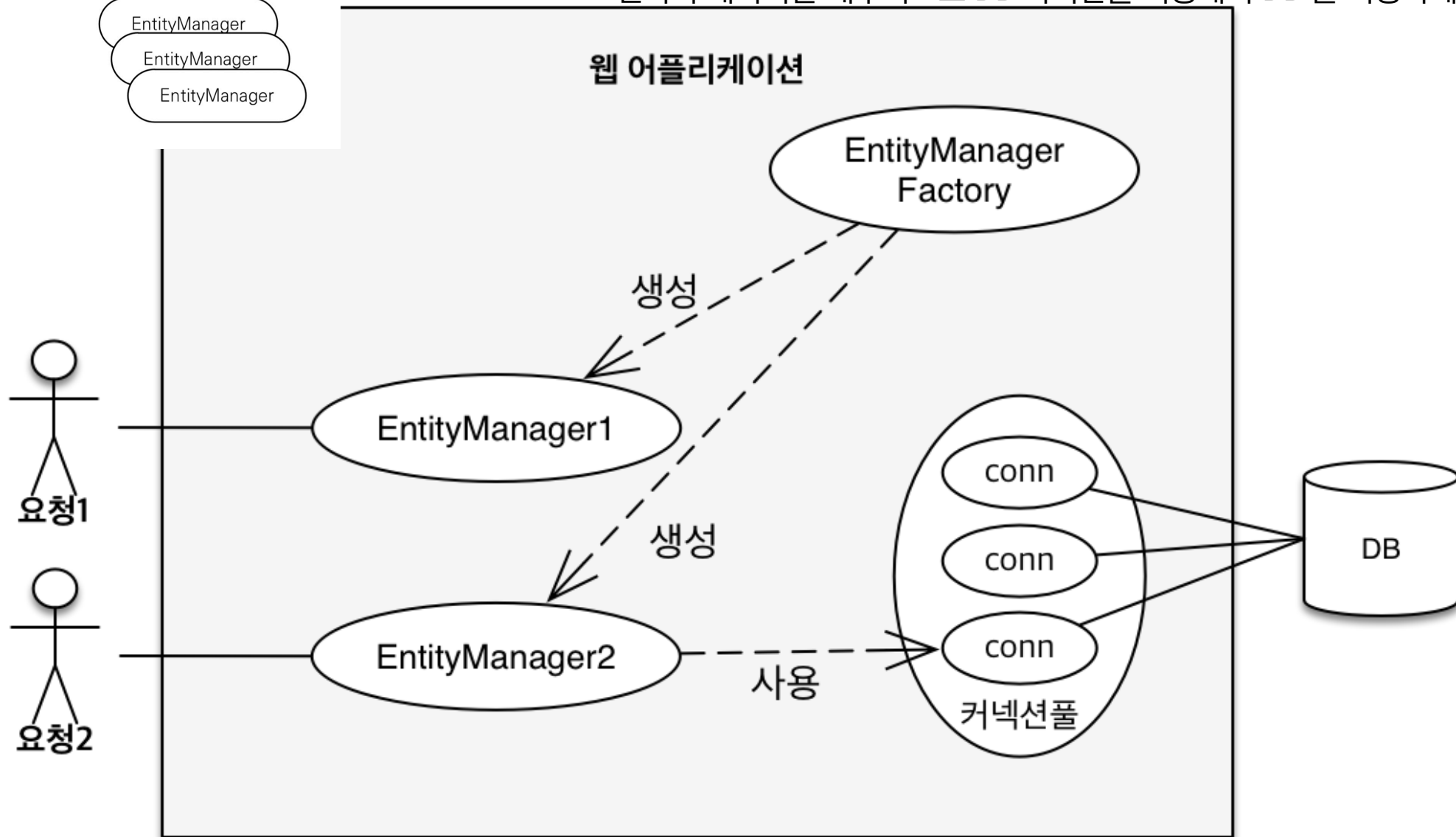
# JPA에서 가장 중요한 2가지

- 객체와 관계형 데이터베이스 매핑하기  
(Object Relational Mapping)
- **영속성 컨텍스트**      실제 JPA 가 내부에서 어떻게 동작하는가?

# 엔티티 매니저 팩토리와 엔티티 매니저



엔티티 매니저 팩토리를 통해서 고객의 요청이 올때마다 엔티티 매니저를 생성하고  
엔티티 매니저는 내부적으로 DB 커넥션을 이용해서 DB 를 이용하게 됨



# 영속성 컨텍스트

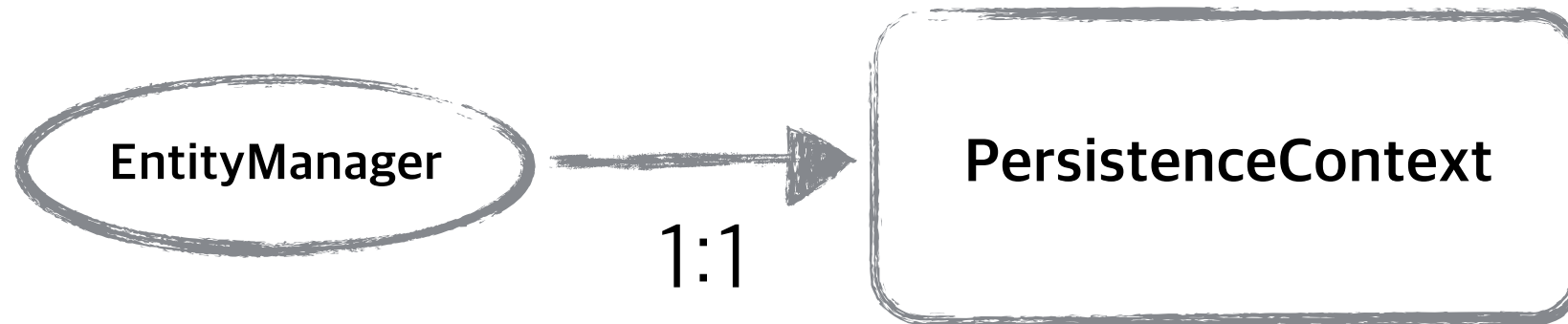
- JPA를 이해하는데 가장 중요한 용어
- “엔티티를 영구 저장하는 환경”이라는 뜻
- **EntityManager.persist(entity);** 엔티티 매니저에서 persist 를 통해서 entity 를 집어넣게 되면 엔티티를 영속성 컨텍스트 안에 집어 넣는다

# 엔티티 매니저? 영속성 컨텍스트?

- 영속성 컨텍스트는 논리적인 개념
- 눈에 보이지 않는다.
- 엔티티 매니저를 통해서 영속성 컨텍스트에 접근

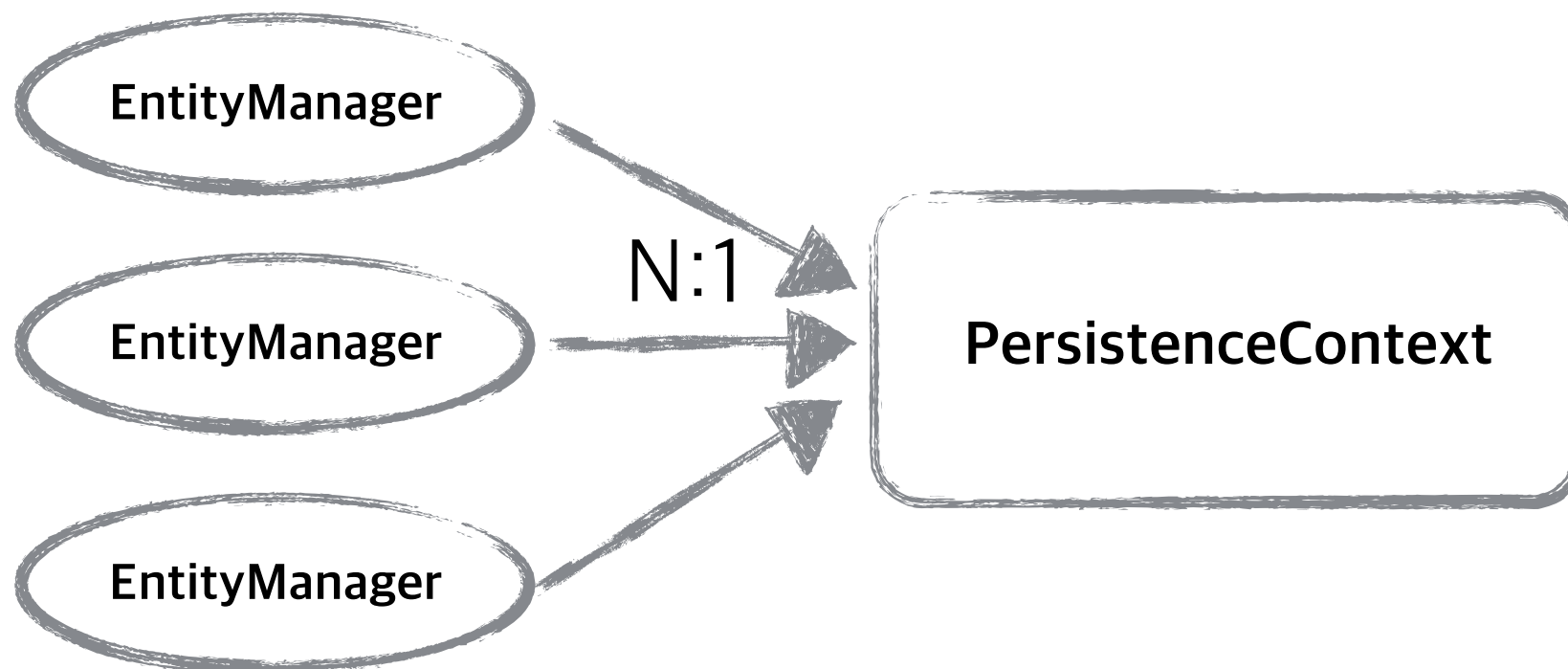
# J2SE 환경

엔티티 매니저와 영속성 컨텍스트가 1:1



# J2EE, 스프링 프레임워크 같은 컨테이너 환경

엔티티 매니저와 영속성 컨텍스트가 N:1





# 엔티티의 생명주기

- **비영속 (new/transient)**

영속성 컨텍스트와 전혀 관계가 없는 새로운 상태

- **영속 (managed)**

영속성 컨텍스트에 관리되는 상태

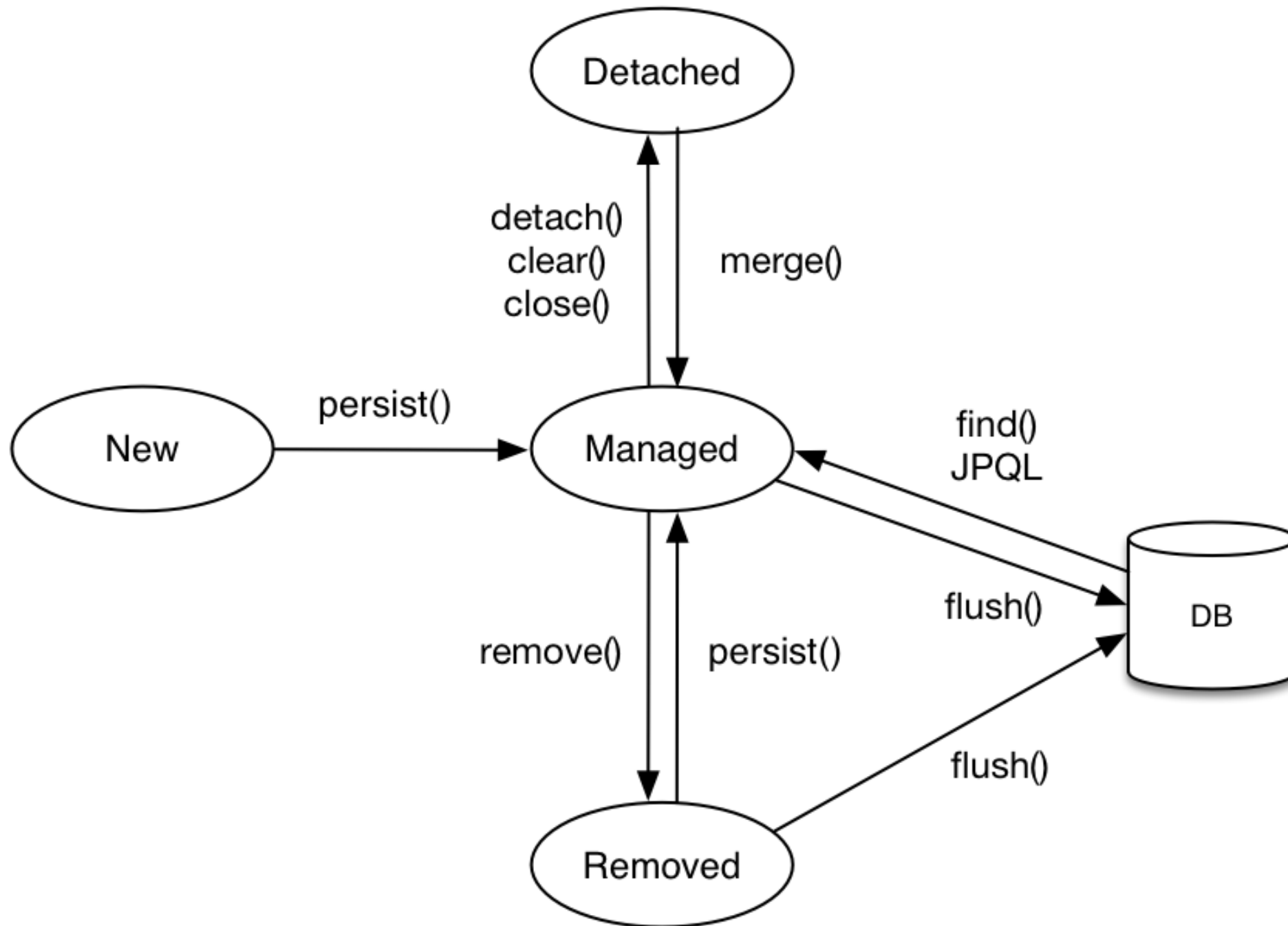
- **준영속 (detached)**

영속성 컨텍스트에 저장되었다가 분리된 상태

- **삭제 (removed)**

삭제된 상태

# 엔티티의 생명주기



# 비영속



영속 컨텍스트(entityManager)

```
//객체를 생성한 상태(비영속)  
Member member = new Member();  
member.setId("member1");  
member.setUsername("회원1");
```

JPA 와 관계 없이 객체만 생성해 둔 상태 - 비영속

# 영속



```
//객체를 생성한 상태 (비영속)  
Member member = new Member();  
member.setId("member1");  
member.setUsername("회원1");
```

```
EntityManager em = emf.createEntityManager();  
em.getTransaction().begin();
```

```
//객체를 저장한 상태 (영속)  
em.persist(member);
```

```
//영속  
System.out.println("=== BEFORE ===");  
em.persist(member);  
System.out.println("=== AFTER ===");
```

```
INFO: HHH000115: Hibernate co  
Jun 01, 2019 12:04:50 AM org.  
INFO: HHH000400: Using dialect  
=== BEFORE ===  
=== AFTER ===  
Hibernate:  
/*insert hellojpa.Member  
*/ insert
```

영속 상태가 된다고 해서 바로 DB에 쿼리가 날아가는게 아님  
em.persist(member); 이후 바로 insert 된게 아님

tx.commit(); 이 되는 순간 DB 에 쿼리가 날아가는 것

# 준영속, 삭제

```
//회원 엔티티를 영속성 컨텍스트에서 분리, 준영속 상태  
em.detach(member);
```

```
//객체를 삭제한 상태(삭제)  
em.remove(member);
```

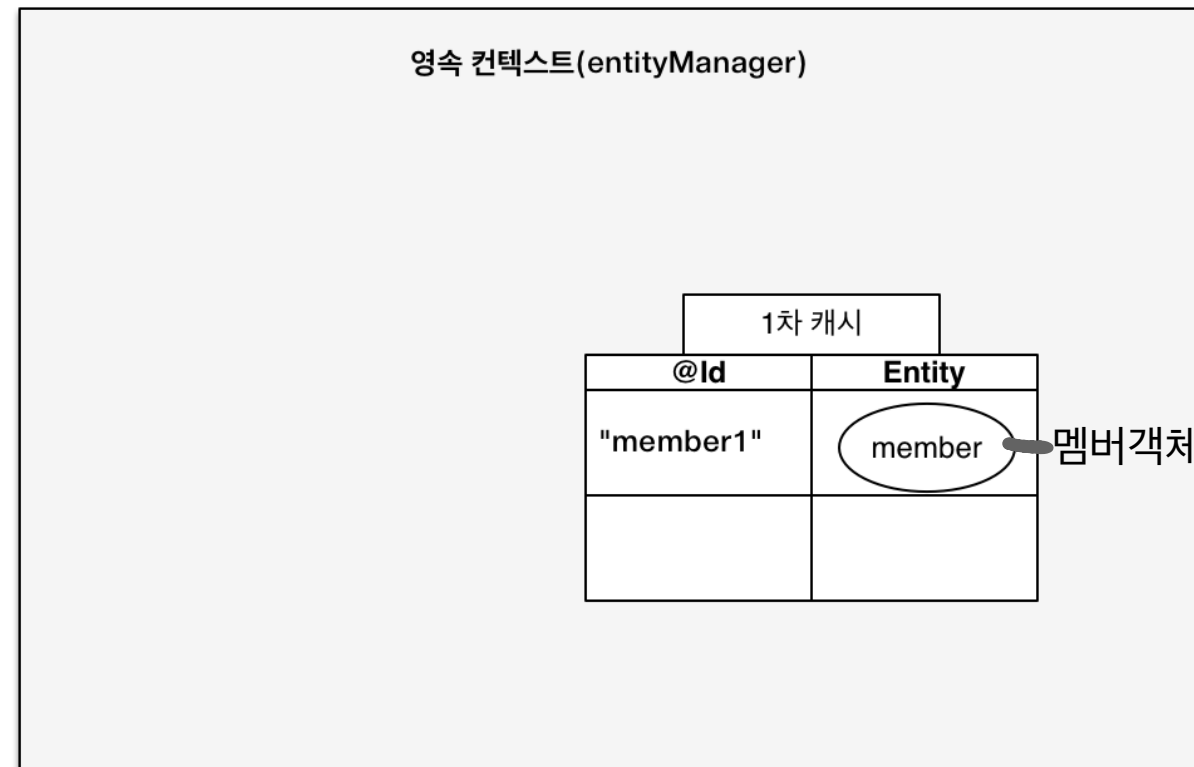
# 영속성 컨텍스트의 이점

영속성 컨텍스트가 필요한 이유

- 1차 캐시
- 동일성(identity) 보장
- 트랜잭션을 지원하는 쓰기 지연  
(transactional write-behind)
- 변경 감지(Dirty Checking)
- 지연 로딩(Lazy Loading)

영속성 컨텍스트의 내부에 1차 캐시가 구현되어 있다.

# 엔티티 조회, 1차 캐시



```
//엔티티를 생성한 상태(비영속)
Member member = new Member();
member.setId("member1");
member.setUsername("회원1");

//엔티티를 영속
em.persist(member);
```

# 1차 캐시에서 조회

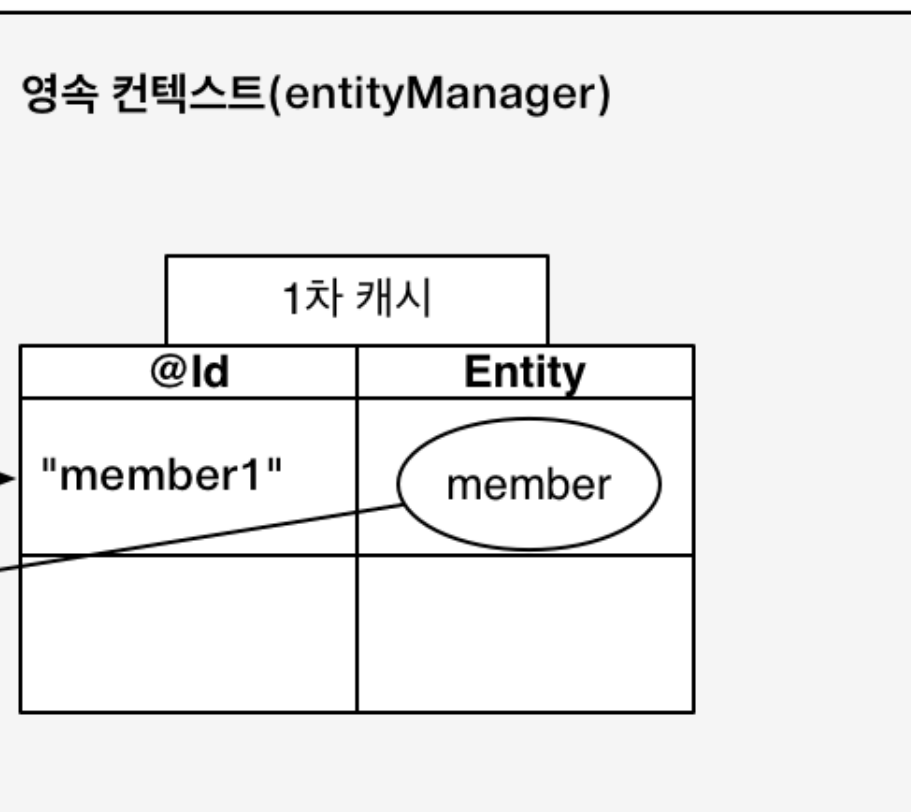
```
Member member = new Member();  
member.setId("member1");  
member.setUsername("회원1");  
  
//1차 캐시에 저장됨  
em.persist(member);  
  
//1차 캐시에서 조회  
Member findMember = em.find(Member.class, "member1");
```

em.find() 를 하면 DB 를 보는게 아니라  
1차 캐시 테이블을 먼저 찾아본다

find("member1")



캐시값 조회





```
//비영속
Member member = new Member();
member.setId(101L);
member.setName("HelloJPA");

//영속
System.out.println("=== BEFORE ===");
em.persist(member);
System.out.println("=== AFTER ===");

Member findMember = em.find(Member.class, primaryKey: 101L);

System.out.println("findMember.id = " + findMember.getId());
System.out.println("findMember.name = " + findMember.getName());
```

```

Sun 01, 2019 12:10:33 AM GMT+09:00
INFO: HHH000400: Using dialect:
=== BEFORE ===
=== AFTER ===
findMember.id = 101
findMember.name = HelloJPA
Hibernate:
        /* insert hellojpa.Member
        */ insert
        into

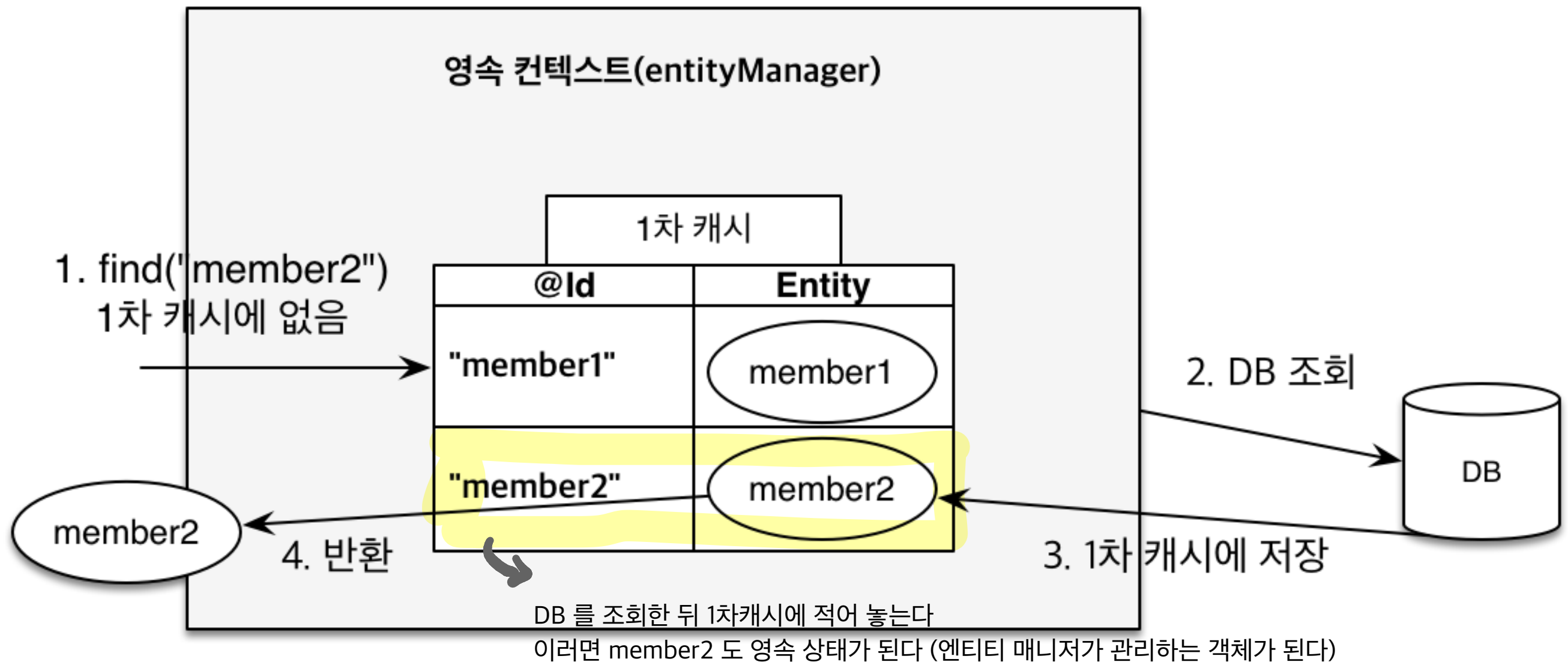
```

HelloJPA 멤버를 찾아올 때  
 Select 로 찾아오지 않음  
 (1차 캐시를 이용한 상황)  
 (persist() 될 때 1차 캐시에 저장된다)

1차 캐시에 없는 member2 를 조회하는 경우

# 데이터베이스에서 조회

```
Member findMember2 = em.find(Member.class, "member2");
```



사실 1차 캐시는 한 트랜잭션 안에서만 효과가 있기 때문에 성능상 눈에 띄는 큰 이점은 없다...  
(트랜잭션이 닫히면 1차 캐시도 날라가는 것)

```
//영속
Member findMember1 = em.find(Member.class, primaryKey: 101L);
Member findMember2 = em.find(Member.class, primaryKey: 101L);
```

```
Jun 01, 2019 12:18:41 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate:
select
    member0_.id as id1_0_0_,
    member0_.name as name2_0_0_
from
    Member member0_
where
    member0_.id=?
Jun 01, 2019 12:18:42 AM org.hibernate.engine.jdbc.connections.inte
INFO: HHH10001008: Cleaning up connection pool [jdbc:h2:tcp://local
```

Select 쿼리가 한번만 날아간다

처음에 가지고 올 때 DB 에서 가지고 온 뒤 영속성 컨텍스트에 올린다.

두번째 find() 시 영속성 컨텍스트 안의 1차 캐시에서 객체를 가지고 오기 때문에 두번째때는 쿼리가 날아가지 않는 것

# 영속 엔티티의 동일성 보장

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

1차 캐시로 반복 가능한 읽기(REPEATABLE READ) 등급의 트랜잭션 격리 수준을 데이터베이스가 아닌 애플리케이션 차원에서 제공

같은 트랜잭션 안에서 == 비교 하면 true 가 나온다!

# 엔티티 등록

## 트랜잭션을 지원하는 쓰기 지연

```
EntityManager em = emf.createEntityManager();  
EntityTransaction transaction = em.getTransaction();  
//엔티티 매니저는 데이터 변경시 트랜잭션을 시작해야 한다.  
transaction.begin(); // [트랜잭션] 시작
```

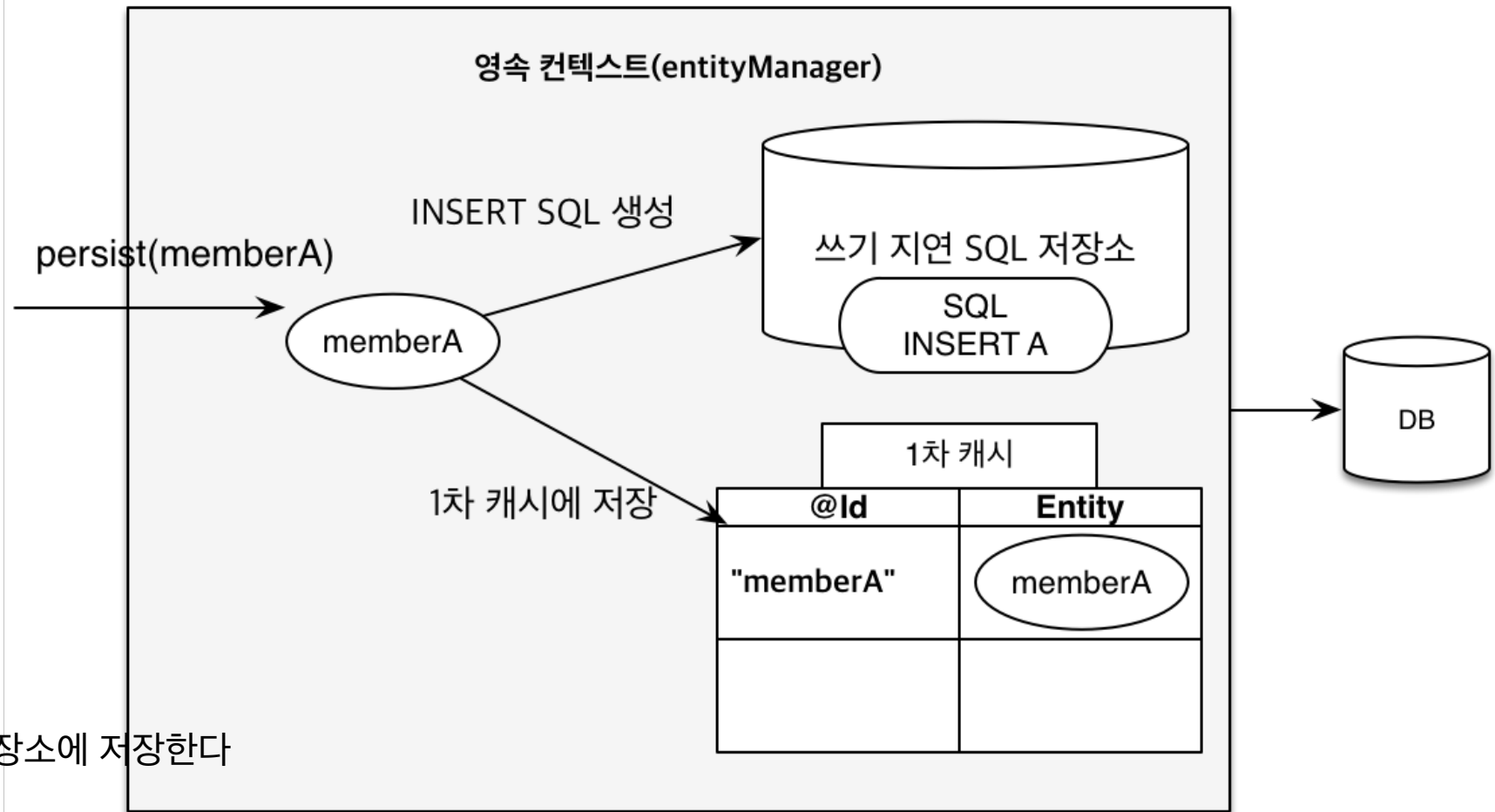
```
em.persist(memberA);  
em.persist(memberB);  
//여기까지 INSERT SQL을 데이터베이스에 보내지 않는다.
```

그냥 persist() 를 할 때마다 짹 짹 쌓아놓고 commit 시점에 필요한 쿼리만 보낸다

```
//커밋하는 순간 데이터베이스에 INSERT SQL을 보낸다.  
transaction.commit(); // [트랜잭션] 커밋
```

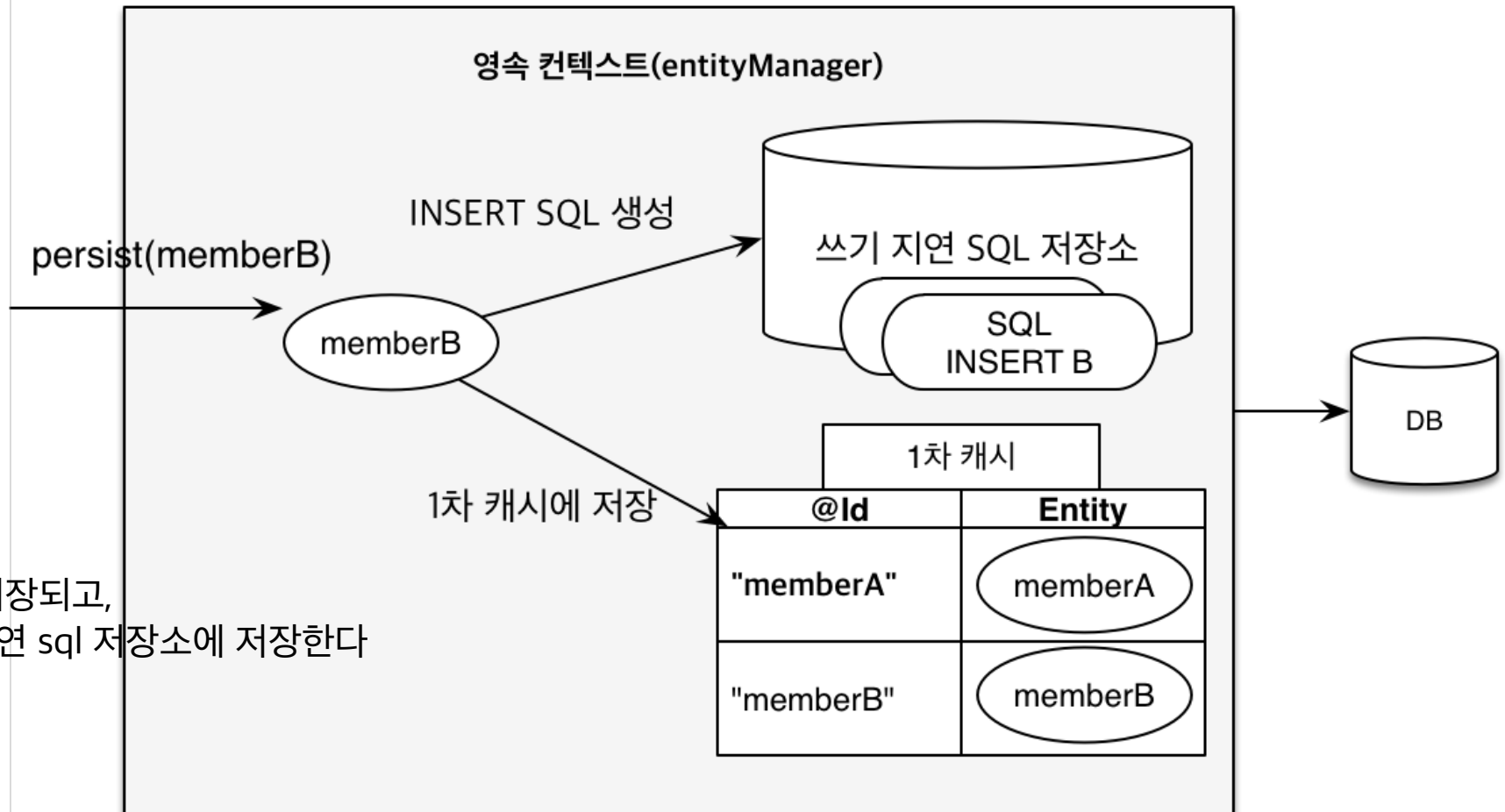
**em.persist(memberA);**

persist(memberA) 하면 1차캐시에 저장하고,  
적당한 insert sql 을 만들어서 쓰기 지연 sql 저장소에 저장한다

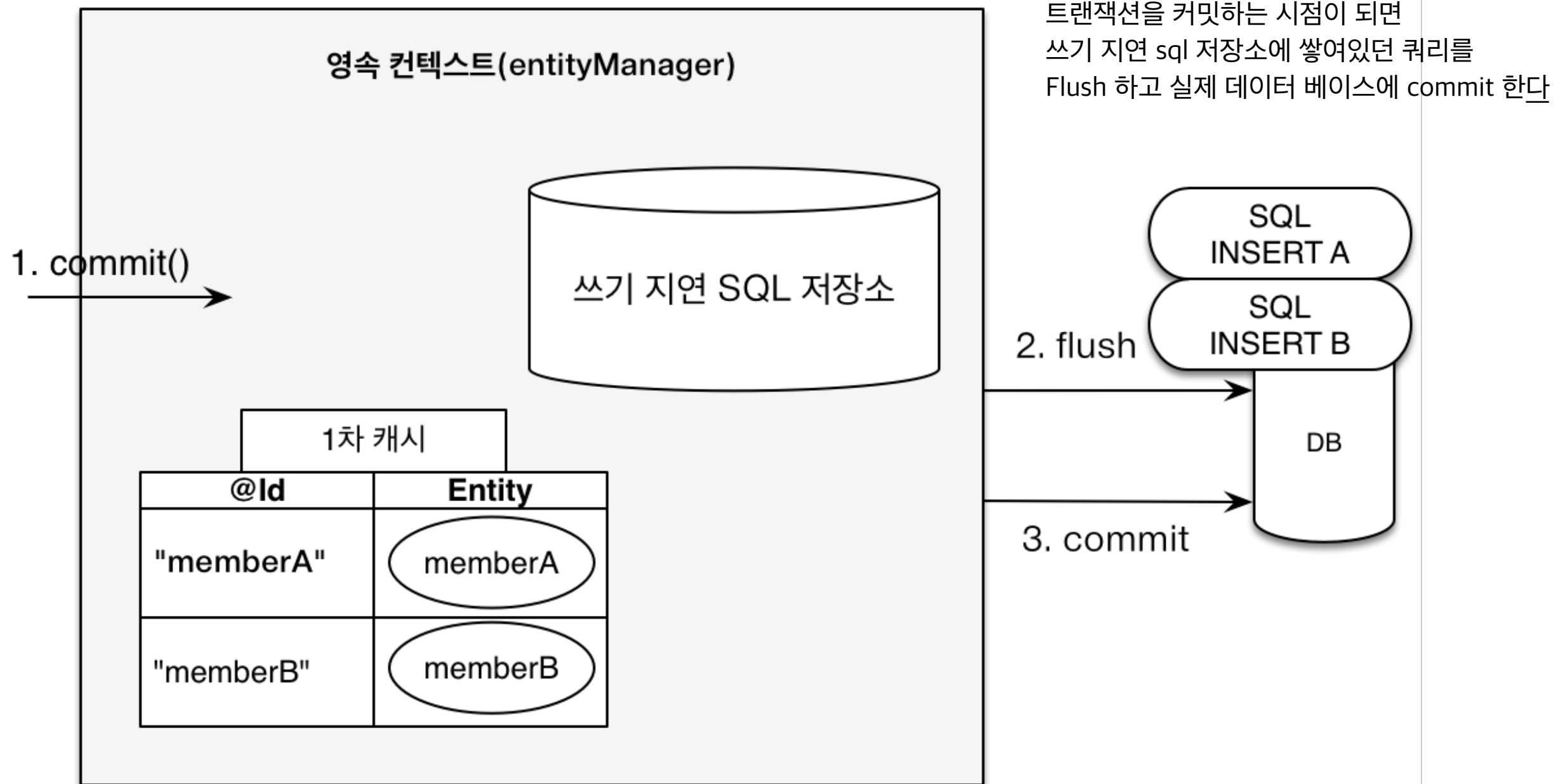


**em.persist(memberB);**

persist(memberB) 하면 똑같이 1차캐시에 저장되고,  
(아까와 같은) insert sql 을 만들어서 쓰기 지연 sql 저장소에 저장한다  
아직까지는 아무 쿼리도 날리지 않은 상태



# transaction.commit();



# 엔티티 수정 변경 감지

```
EntityManager em = emf.createEntityManager();
EntityTransaction transaction = em.getTransaction();
transaction.begin();    // [트랜잭션] 시작

// 영속 엔티티 조회
Member memberA = em.find(Member.class, "memberA");

// 영속 엔티티 데이터 수정
memberA.setUsername("hi");
memberA.setAge(10);

//em.update(member) 이런 코드가 있어야 하지 않을까?

transaction.commit();    // [트랜잭션] 커밋
```



JPA의 목적은 자바 컬렉션 다루듯이 객체를 다루는 것  
리스트에서 값을 꺼내고 객체의 값을 수정하면 다시 리스트에 넣느냐? ㅋㅋ 그냥 꺼내서 수정하잖아  
그렇게 JPA 를 쓰면 된다

```
Member member = em.find(Member.class, primaryKey: 150L);  
member.setName("ZZZZZ");  
em.persist(member);  
System.out.println("=====");  
tx.commit();
```

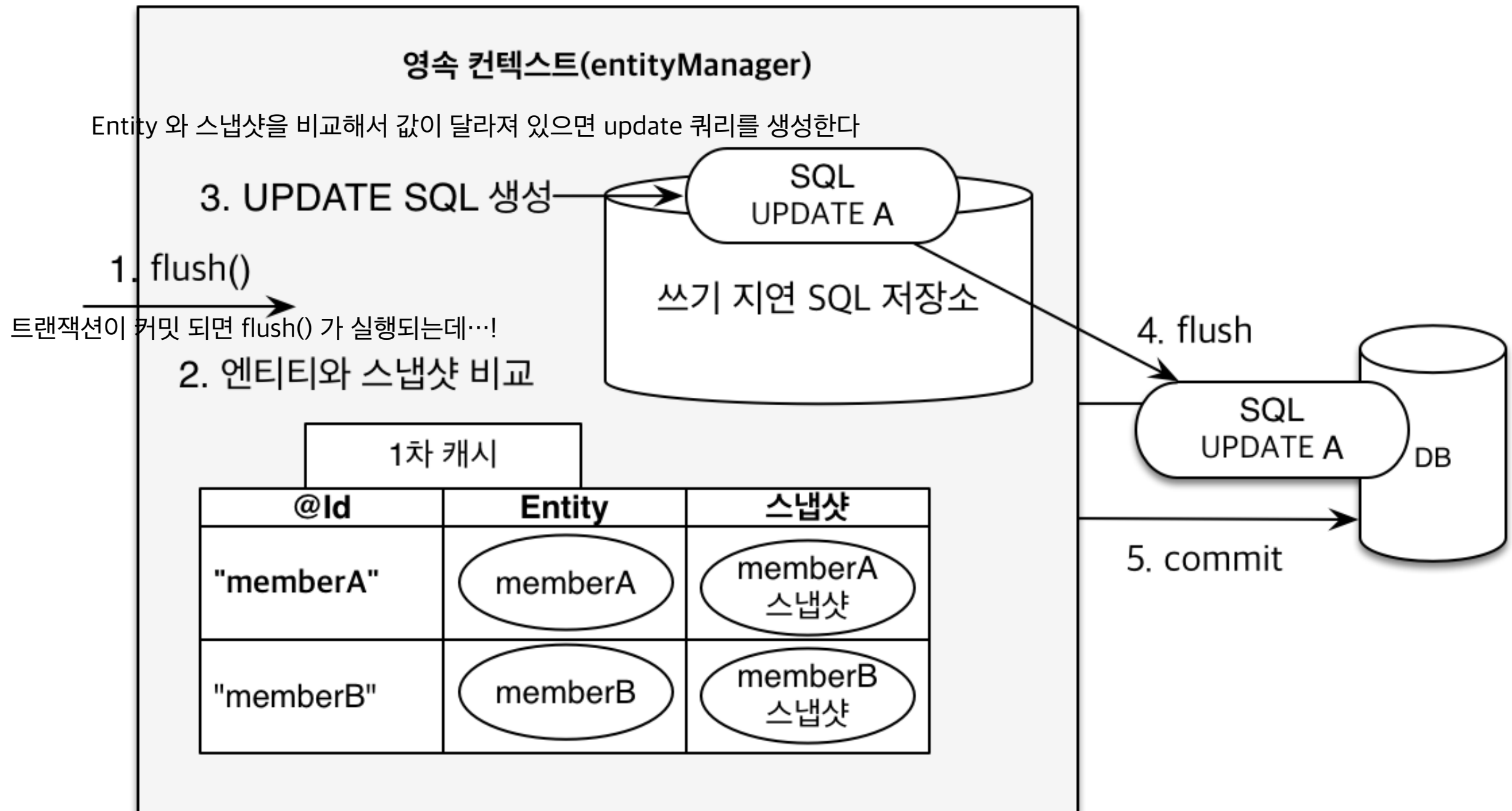
```
Jun 01, 2019 12:33:43 AM org.hibernate.  
INFO: HHH000400: Using dialect: org.hib  
Hibernate:  
  select  
    member0_.id as id1_0_0_,  
    member0_.name as name2_0_0_  
  from  
    Member member0_  
  where  
    member0_.id=?  
=====
```

```
Hibernate:  
  /* update  
    hellojpa.Member */ update
```

Select 날아가고  
print() 실행 되고  
Update 가 된다

굳이 persist() 를 날려줄 필요가 없다

# 변경 감지 (Dirty Checking)



스냅샷? DB에서 처음 읽어왔던 시점의 객체

# 엔티티 삭제

//삭제 대상 엔티티 조회

```
Member memberA = em.find(Member.class, "memberA");
```

```
em.remove(memberA); //엔티티 삭제
```

트랜잭션 커밋 시점에 delete 쿼리가 나간다

# 플러스

영속성 컨텍스트의 변경내용을  
데이터베이스에 반영

데이터베이스가 커밋 되면 플러시가 발생한다

# 플러시 발생

- 변경 감지
- 수정된 엔티티 쓰기 지연 SQL 저장소에 등록
- 쓰기 지연 SQL 저장소의 쿼리를 데이터베이스에 전송  
(등록, 수정, 삭제 쿼리)

# 영속성 컨텍스트를 플러시하는 방법

- `em.flush()` - 직접 호출
- **트랜잭션 커밋** - 플러시 자동 호출
- **JPQL 쿼리 실행** - 플러시 자동 호출

```

Member member = new Member( id: 200L, name: "member200");
em.persist(member);

em.flush();

System.out.println("=====");
tx.commit();

```

```

Jun 01, 2019 12:44:24 AM org.hibernate
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
Hibernate:
    /* insert hellojpa.Member
       */ insert
       into
           Member
           (name, id)
       values
           (?, ?)
=====
Jun 01, 2019 12:44:25 AM org.hibernate
INFO: HHH10001008: Cleaning up connection pool

```

println("=====") 이 되기 전에  
Insert 쿼리가 날아가고  
tx.commit() 될 때는 아무 쿼리도 날아가지 않음

1차 캐시가 지워지지 않는다는

# JPQL 쿼리 실행시 플러시가 자동 으로 호출되는 이유

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

→ 이때까지 데이터베이스에 날아간 쿼리는 없는 상황

//중간에 JPQL 실행

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

→ But! JPQL 쿼리를 실행하면 우선 플러시가 자동으로 호출된다.




# 플러시 모드 옵션

```
em.setFlushMode(FlushModeType.COMMIT)
```

- **FlushModeType.AUTO**  
커밋이나 쿼리를 실행할 때 플러시 (기본값)
- **FlushModeType.COMMIT**  
커밋할 때만 플러시 (쿼리를 실행할 땐 플러시하지 않겠다)

# 플러시는!

- 영속성 컨텍스트를 비우지 않음
- 영속성 컨텍스트의 변경내용을 데이터베이스에 동기화 
- 트랜잭션이라는 작업 단위가 중요 -> 커밋 직전에만 동기화 하면 됨

# 준영속 상태

영속 상태 (1) persist (2) find  
즉 1차 캐시에 올라와 있으면 영속 상태이다

- 영속 -> 준영속
- 영속 상태의 엔티티가 영속성 컨텍스트에서 분리(detached)
- 영속성 컨텍스트가 제공하는 기능을 사용 못함

# 준영속 상태로 만드는 방법

- **em.detach(entity)**  
특정 엔티티만 준영속 상태로 전환
- **em.clear()**  
영속성 컨텍스트를 완전히 초기화
- **em.close()**  
영속성 컨텍스트를 종료

```
//영속
Member member = em.find(Member.class, primaryKey: 150L);
member.setName("AAAAA");

em.detach(member);

System.out.println("=====");
tx.commit();
catch (Exception e) {
```

```
INFO: HHH000400: Using dialect: org.hibe
Hibernate:
  select
    member0_.id as id1_0_0_,
    member0_.name as name2_0_0_
  from
    Member member0_
  where
    member0_.id=?
=====
Jun 01, 2019 12:53:10 AM org.hibernate.e
INFO: HHH10001008: Cleaning up connection
Process finished with exit code 0
```

find() 할 때 1차 캐시에 없으니까 DB에서 조회해와서  
영속성 컨텍스트에 올린다  
(1차 캐시에 올려 영속 상태에 둔다.)

그리고 값을 변경했으니 더티체킹 할 것  
setName() 을 했으니 update 쿼리를 만들어서  
SQL 저장소에 저장한 뒤  
트랜잭션이 커밋되면 실제 쿼리를 날릴 예정이었음.

근데 영속성 컨텍스트가 그 전에 detach() 해버리면  
트랜잭션을 커밋할 때 아무 쿼리도 날아가지 않는다  
영속성 컨텍스트에서 빠져버렸기 때문