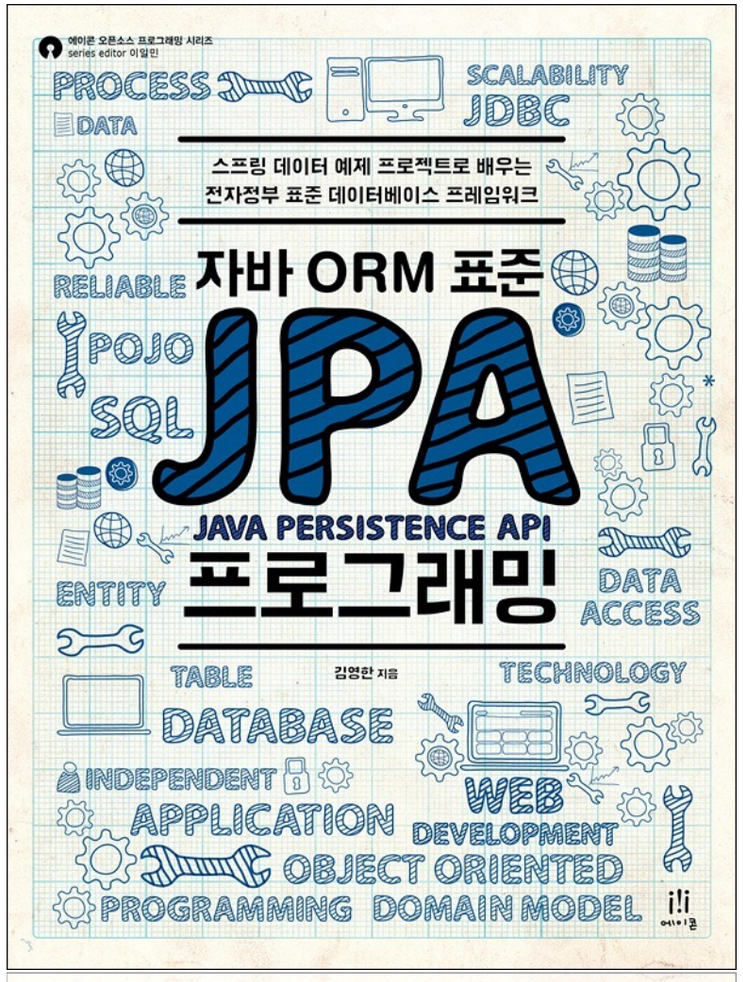
JPA 내부 구조

김영한

SI, J2EE 강사, DAUM, SK 플래닛

저서: 자바 ORM 표준 **JPA** 프로그래밍



목차

- 영속성 컨텍스트
- 프록시와 즉시로딩, 지연로딩

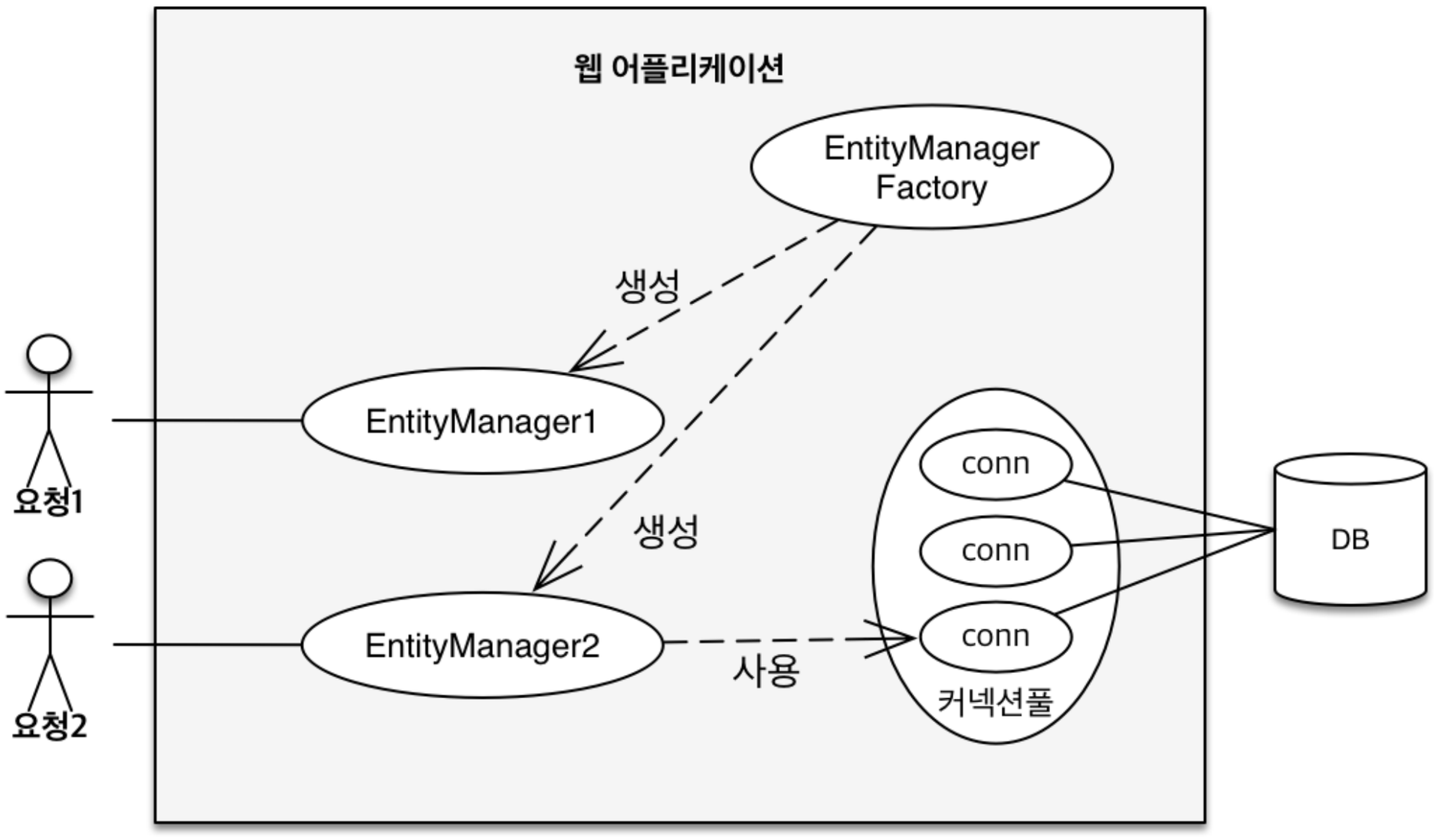
영속성 컨텍스트

JPA에서 가장 중요한 2가지

- 객체와 관계형 데이터베이스 매핑하기
  (Object Relational Mapping)

- **영속성 컨텍스트**

엔티티 매니저 팩토리와 엔티티 매니저



영속성 컨텍스트

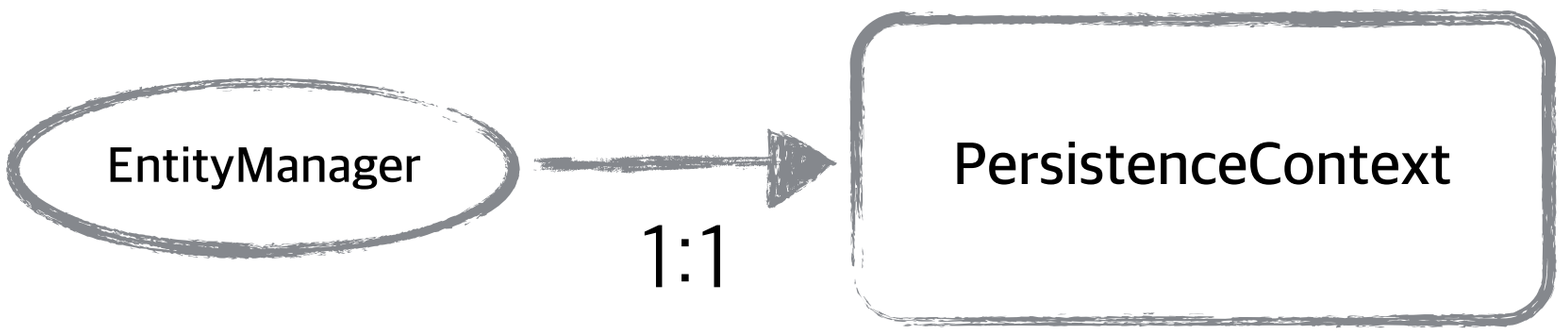
- JPA를 이해하는데 가장 중요한 용어
- “엔티티를 영구 저장하는 환경”이라는 뜻
- **EntityManager.persist(entity);**

엔티티 매니저?
영속성 컨텍스트?

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

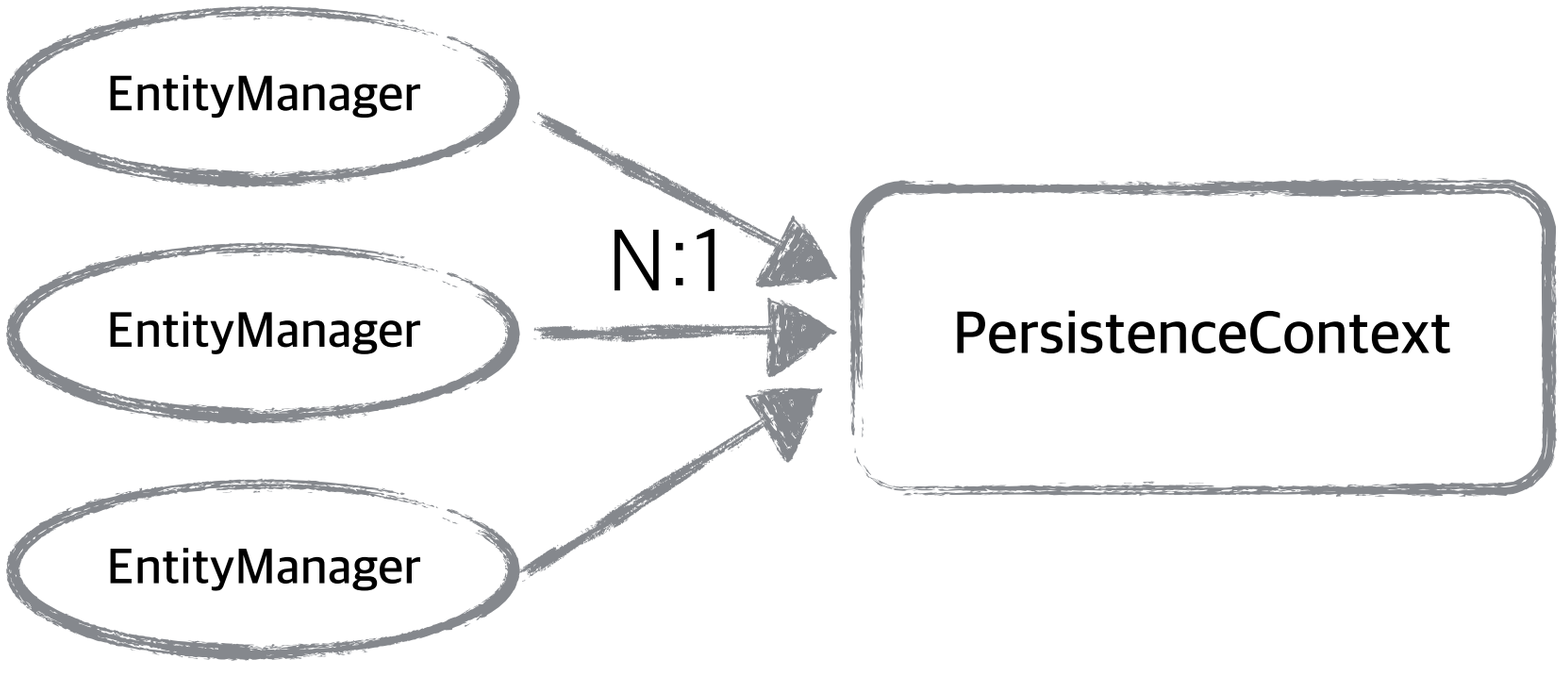
J2SE 환경

엔티티 매니저와 영속성 컨텍스트가 1:1



J2EE, 스프링 프레임워크 같은 컨테이너 환경

엔티티 매니저와 영속성 컨텍스트가 N:1



엔티티의 생명주기

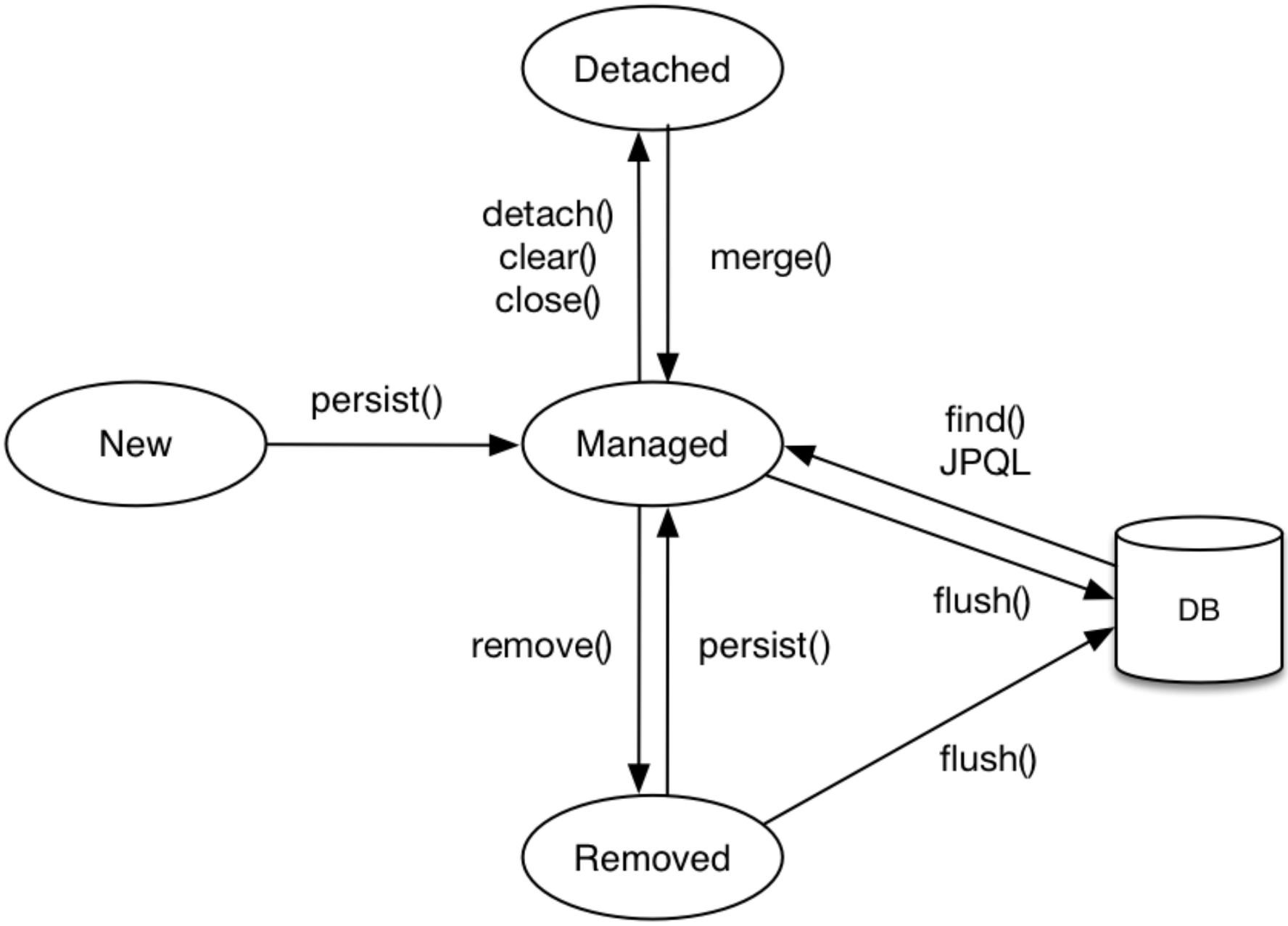
- **비영속 (new/transient)**
  영속성 컨텍스트와 전혀 관계가 없는 상태

- **영속 (managed)**
  영속성 컨텍스트에 저장된 상태

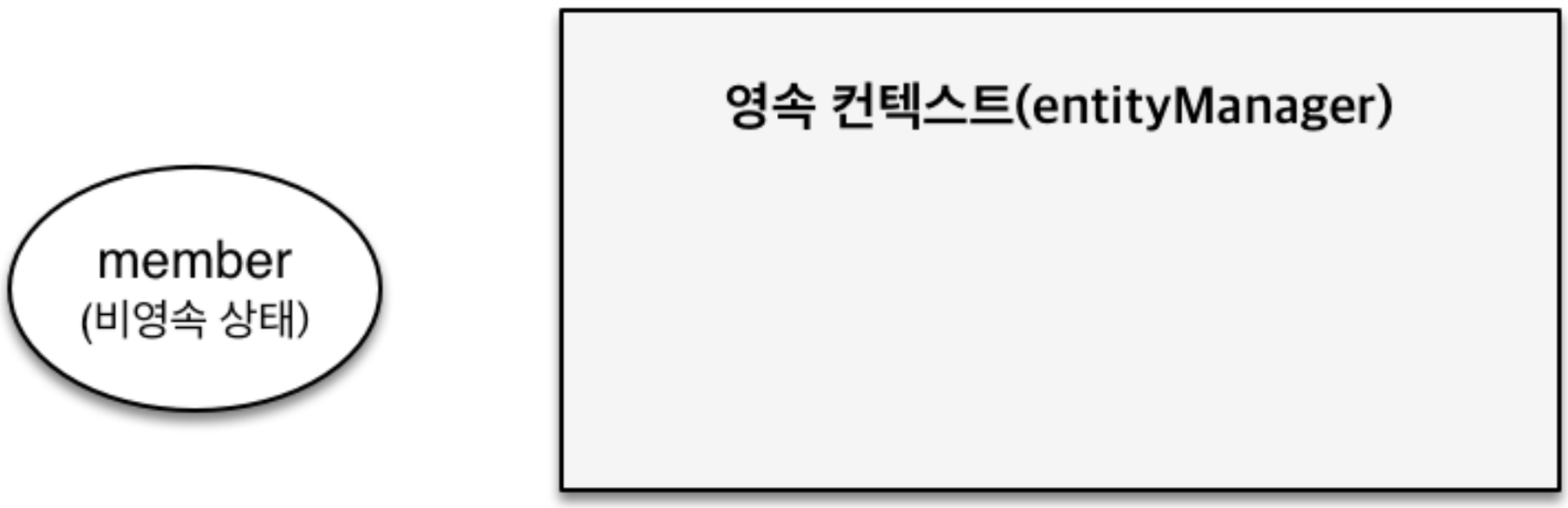
- **준영속 (detached)**
  영속성 컨텍스트에 저장되었다가 분리된 상태

- **삭제 (removed)**
  삭제된 상태

엔티티의 생명주기


Detached
detach()
clear()
close()
merge()
New
persist()
Managed
find()
JPQL
flush()
DB
remove()
persist()
flush()
Removed

비영속




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

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);
```

준영속, 삭제

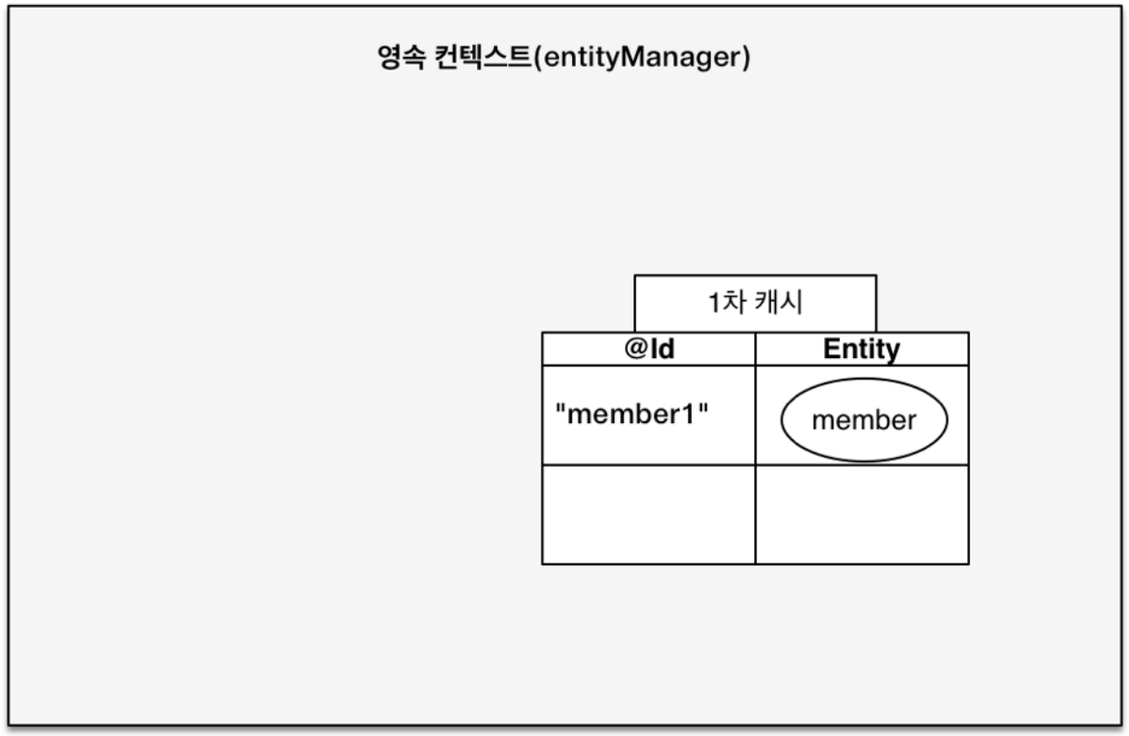
```
//회원 엔티티를 영속성 컨텍스트에서 분리, 준영속 상태
em.detach(member);
```

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

영속성 컨텍스트의 이점

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

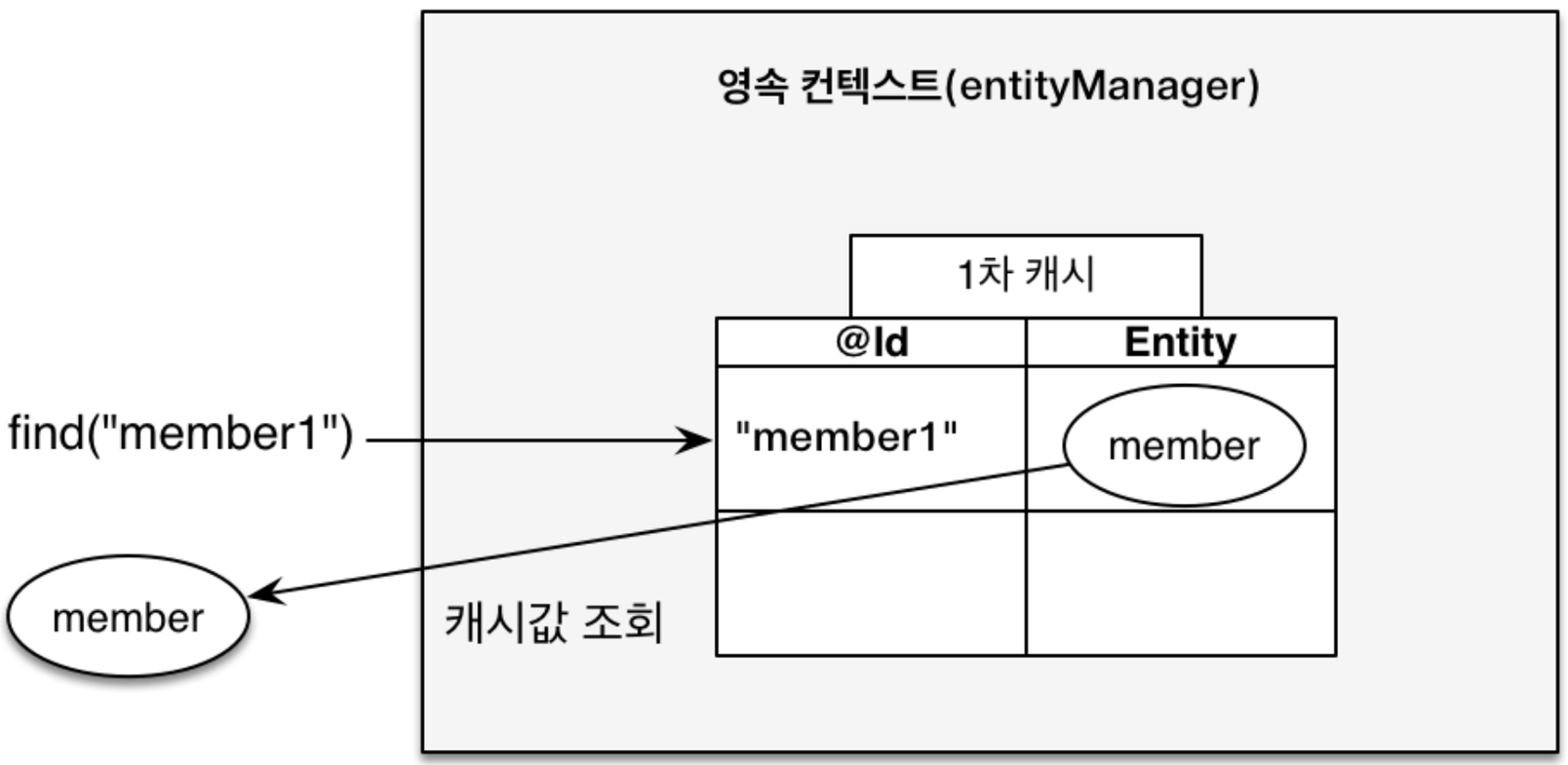
엔티티 조회, 1차 캐시




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

//엔티티를 영속
em.persist(member);
```

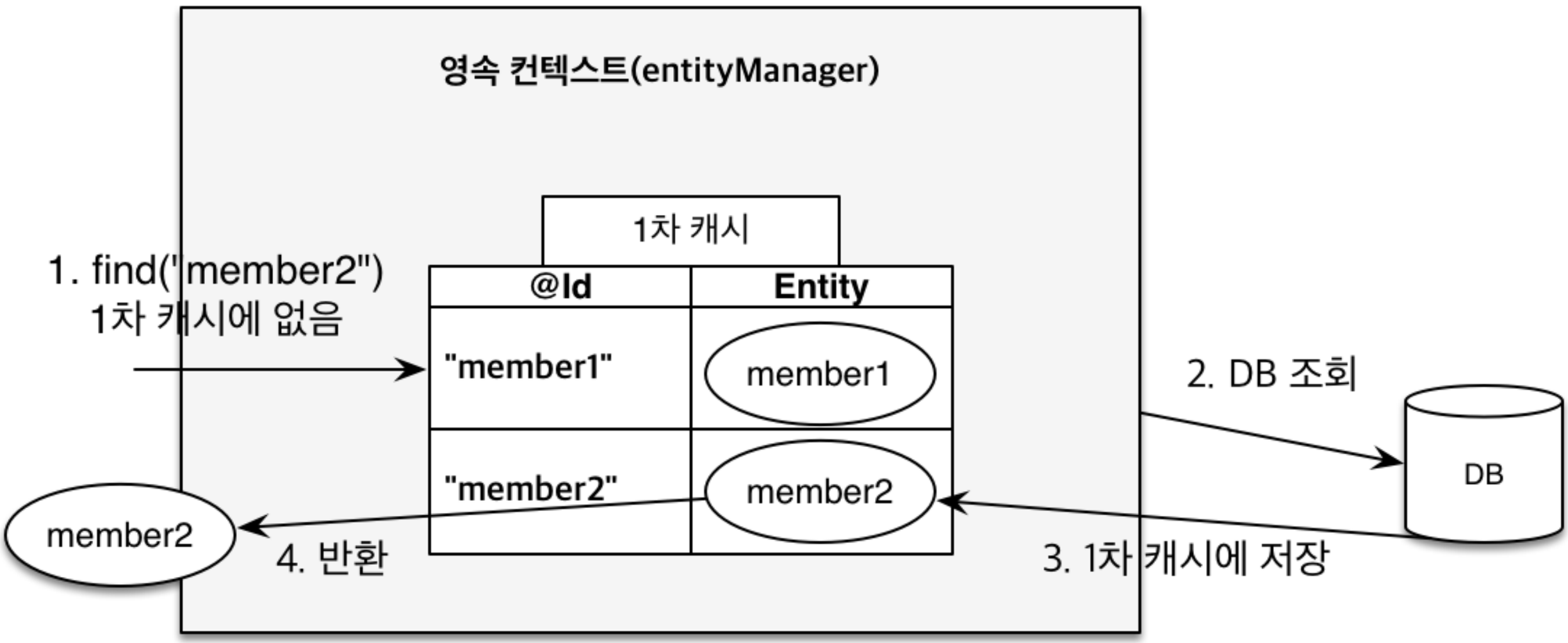
1차 캐시에서 조회

```
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");
```



데이터베이스에서 조회

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



영속 엔티티의 동일성 보장

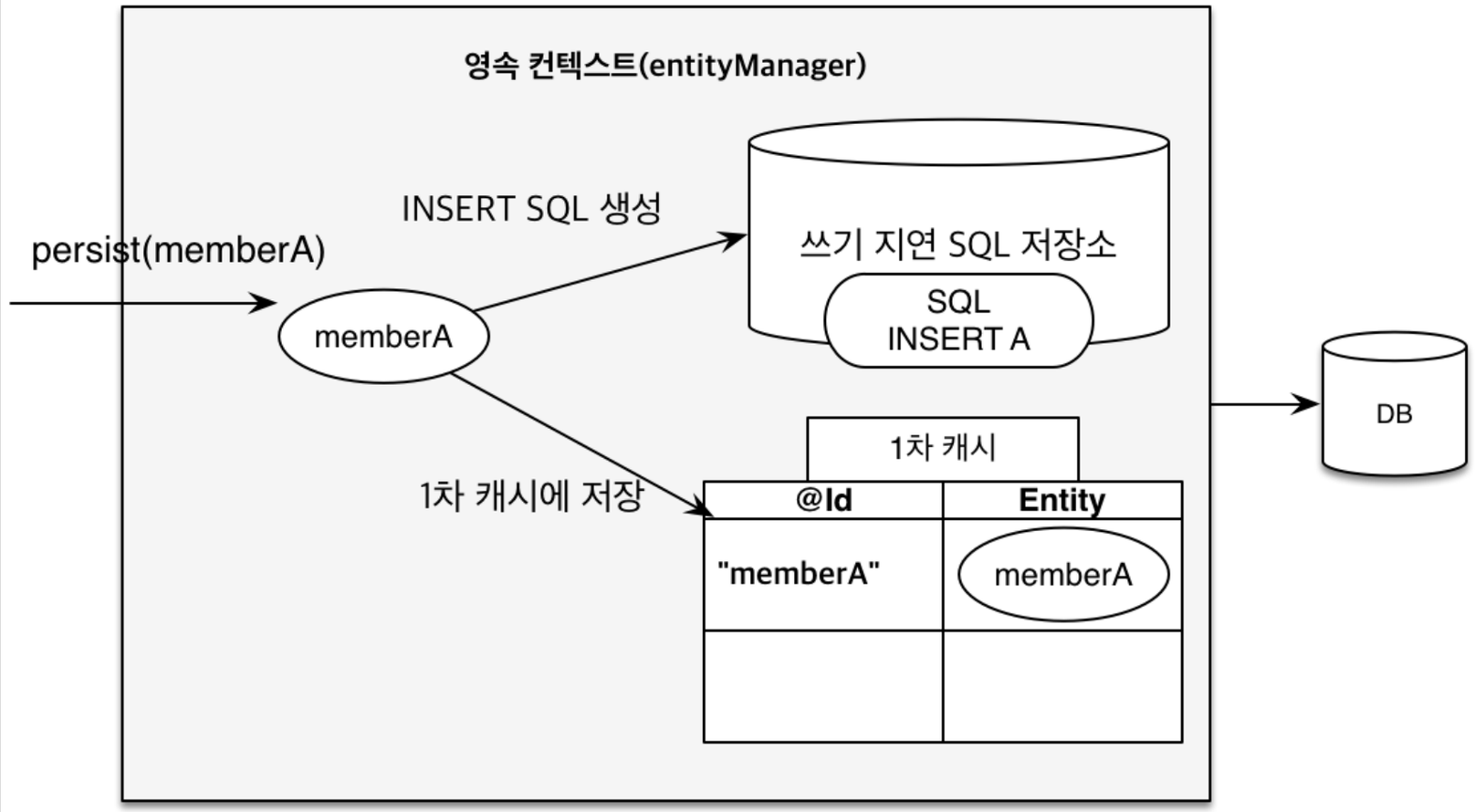
```
Member a = em.find(Member.class, "member1");
Member b = em.find(Member.class, "member1");

System.out.println(a == b); //동일성 비교 true
```

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

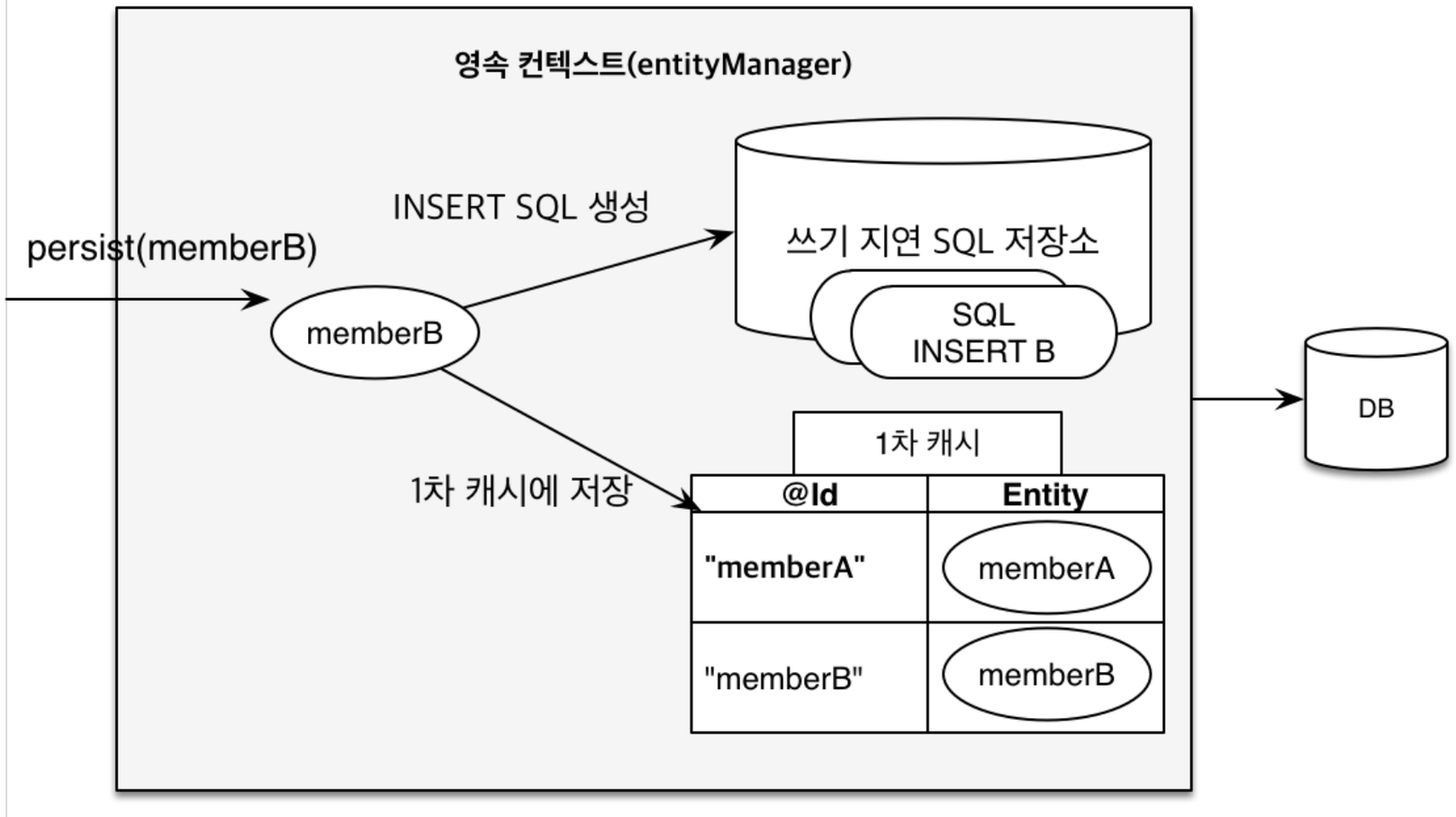
엔티티 등록
트랜잭션을 지원하는 쓰기 지연

```
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.

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

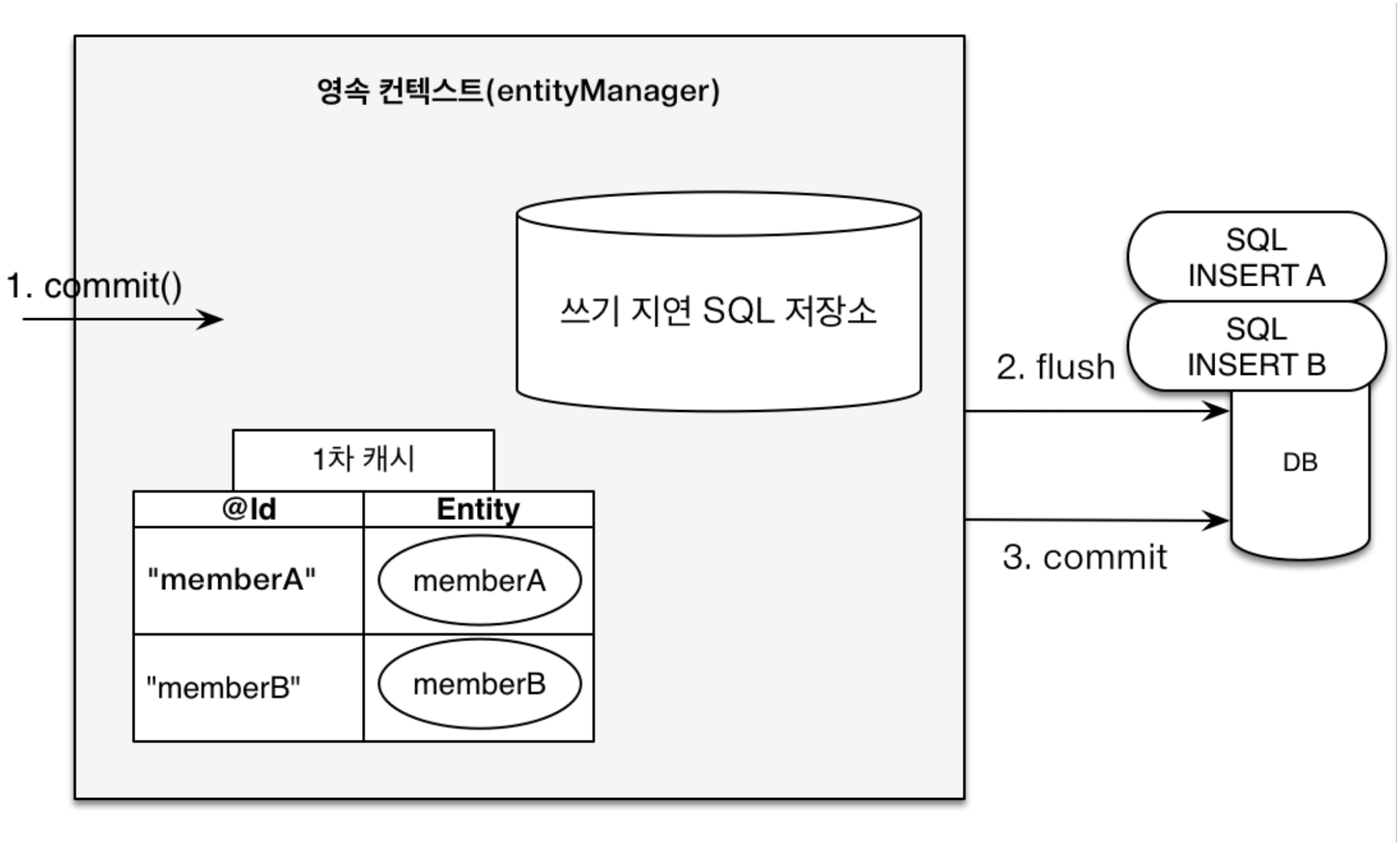
```
em.persist(memberA);
```




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



transaction.commit();

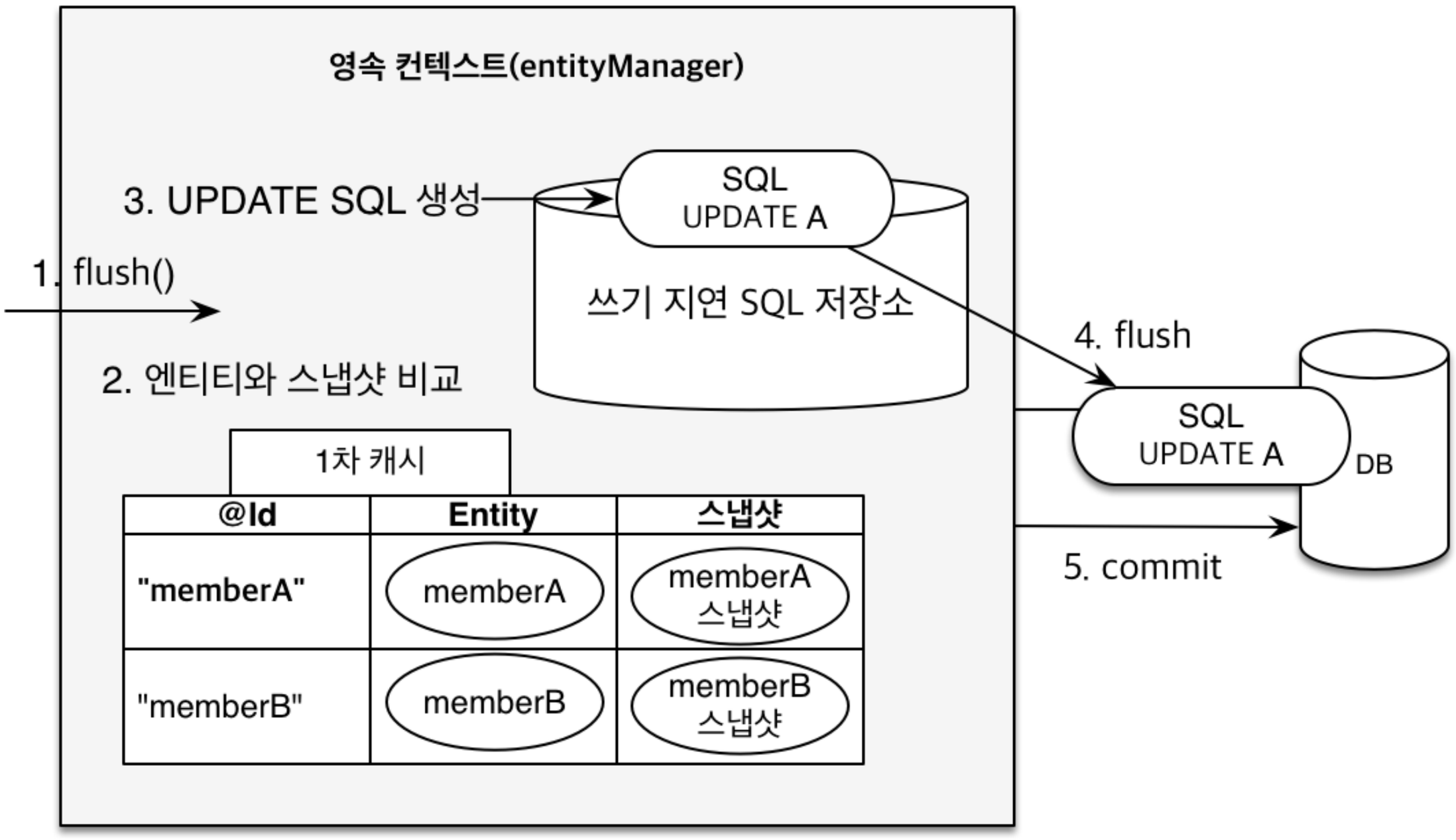


엔티티 수정
변경 감지

```
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
```

변경 감지

(Dirty Checking)



엔티티 삭제

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

em.remove(memberA); //엔티티 삭제
```

플러시

영속성 컨텍스트의 변경내용을
데이터베이스에 반영

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

JPQL 쿼리 실행시 플러시가 자동으로 호출되는 이유

```
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();
```

플러시 모드 옵션

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

- FlushModeType.AUTO
  커밋이나 쿼리를 실행할 때 플러시 (기본값)

- **FlushModeType.COMMIT**
  커밋할 때만 플러시

플러시는!

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

준영속 상태

- 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

프록시와 즉시로딩, 지연로딩

Member를 조회할 때 Team도 함께 조회해야 할까?

단순히 member 정보만 사용하는 비즈니스 로직

println(member.getName());

지연 로딩 LAZY을 사용해서 프록시로 조회

```
@Entity
public class Member {

  @Id
  @GeneratedValue
  private Long id;

  @Column(name = "USERNAME")
  private String name;

  @ManyToOne(fetch = FetchType.LAZY) //**
  @JoinColumn(name = "TEAM_ID")
  private Team team;
  ..
}
```

지연 로딩 LAZY을 사용해서 프록시로 조회


```
Member member = em.find(Member.class, 1L);
```


```
Team team = member.getTeam();
team.getName(); // 실제 team을 사용하는 시점에 초기화(DB 조회)
```

Member와 Team을 자주 함께 사용한다면?

즉시 로딩 EAGER를 사용해서 함께 조회

```java
@Entity
public class Member {

  @Id
  @GeneratedValue
  private Long id;

  @Column(name = "USERNAME")
  private String name;

  @ManyToOne(fetch = FetchType.EAGER) //**
  @JoinColumn(name = "TEAM_ID")
  private Team team;
  ..
}
```

즉시 로딩(EAGER), Member조회시 항상 Team도 조회

JPA 구현체는 가능하면 조인을 사용해서 SQL 한번에 함께 조회

프록시와 즉시로딩 주의

- **가급적 지연 로딩을 사용**
- 즉시 로딩을 적용하면 예상하지 못한 SQL이 발생
- 즉시 로딩은 JPQL에서 N+1 문제를 일으킨다.
- @ManyToOne, @OneToOne은 기본이 즉시 로딩
  -> LAZY로 설정
- @OneToMany, @ManyToMany는 기본이 지연 로딩

실습 ex06

- 즉시 로딩과 지연 로딩시 쿼리 확인