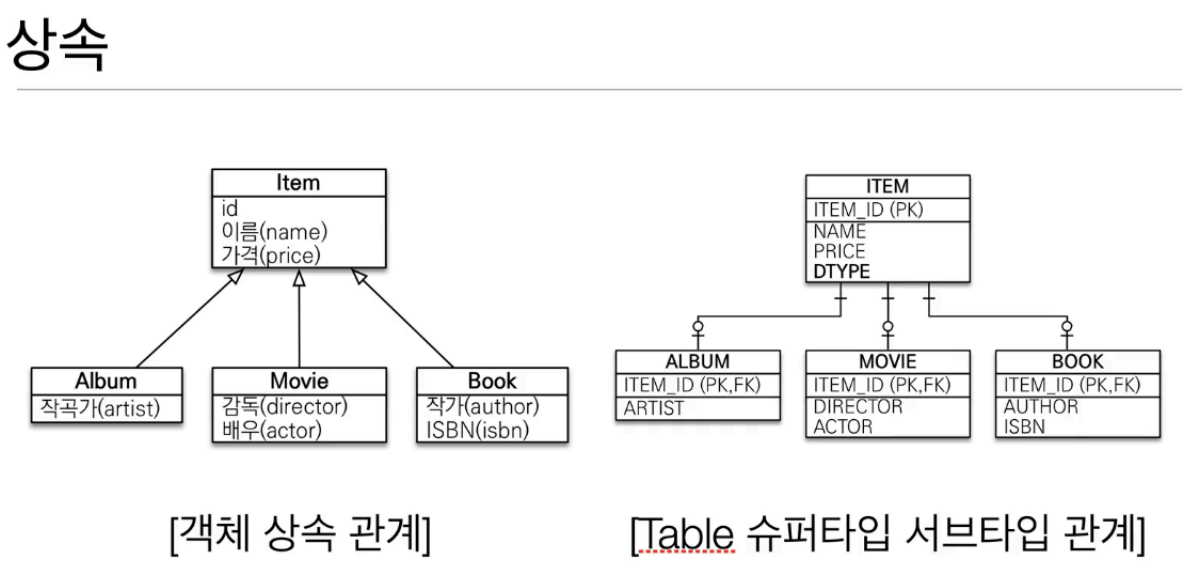
JPA 기초

- 학습 목표
  - 객체와 테이블을 제대로 설계하고 매칭하는 방법
  - 기본 키와 외래 키 매핑
  - 1:N, N:1, 1:1 N:M 매핑
  - 실무 적용 사례 및 성능
  - 복잡한 시스템 JPA 설계
  - JPA 내부 동작 방식 이해
  - JPA 내부 동작 방식을 그림과 코드로 자세히 설명
  - 어떤 SQL , 언제 JPA가 실행되는지

- 왜? JPA 인가
  - SQL 중심으로 개발하게 되면 컬럼의 변경사항이 생기면 객체도 수정해야한다.

- 객체 지향 프로그램밍은 추상화, 캡슐화, 정보은닉, 상속, 다형성 등 복잡성을 제어할 수 있는 다양한 장치들을 제공한다. 즉, 객체를 중심으로 분석 설계하는 방법.

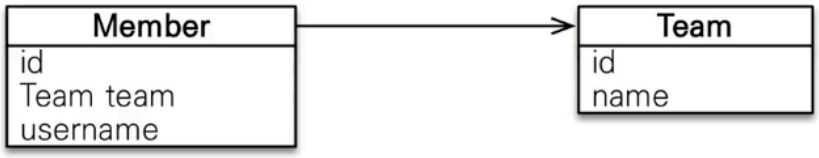
- 객체와 RDB 차이
  - 상속



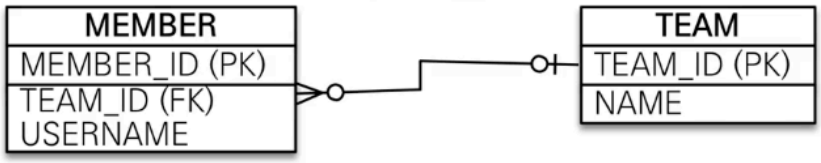
  - 연관관계

- 객체는 **참조**를 사용: member.getTeam()
- 테이블은 **외래 키**를 사용: JOIN ON M.TEAM_ID = T.TEAM_ID

[객체 연관관계]



[테이블 연관관계]

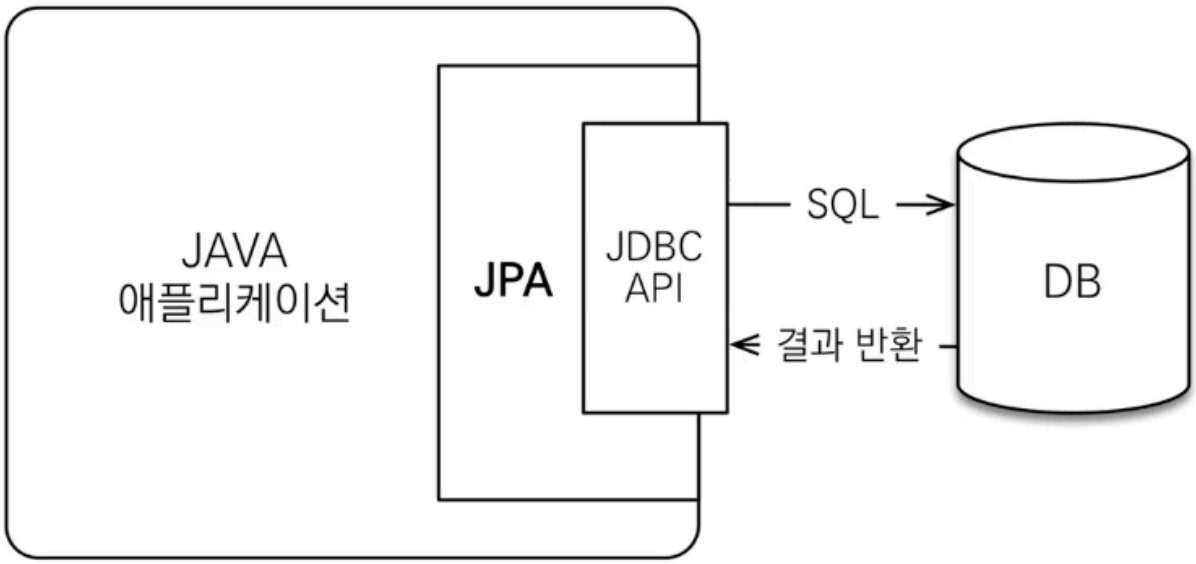


    - 모든 객체를 미리 로딩할 수는 없다.
- 테이터 타입
- 데이터 식별 방법

JPA (Java Persistence API)

Java ORM 기술 표준

ORM (Object-relational mapping) 객체 관계 매핑



- 동일한 트랜잭션에서 조회하는 엔티티는 같음 보장
- 1차 캐시와 동일성 보장
    - [데이터] ➜ [모아서 보내거나 캐시] ➜ [데이터]
    - 같은 트랜잭션 안에서 같은 엔티티 반환

- DB Isolation Level 이 Read Commit 이어도 애플리케이션에서 Repeatable Read 보장

```
String memberId = "100";
Member m1 = jpa.find(Member.class, memberId); //SQL
Member m2 = jpa.find(Member.class, memberId); //캐시

println(m1 == m2) //true
```

SQL 1번만 실행

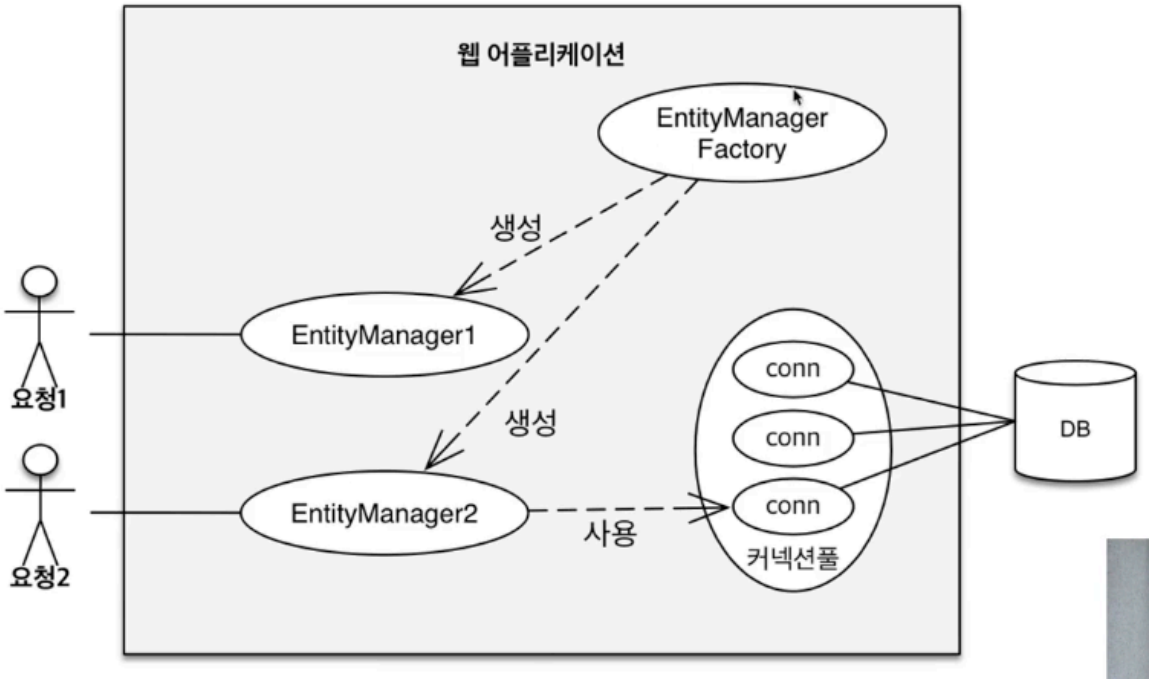
- 트랜잭션을 지원하는 쓰기 지연
  - 커밋할 때까지 insert SQL 모음
  - JDBC BATCH SQL 기능을 사용해서 한번에 전송
  - UPDATE, DELETE 로 인한 로우(ROW) 락 시간 최소화
  - 트랜잭션 커밋 시 UPDATE, DELETE SQL 실행하고 바로 커밋

- 지연 로딩
  - 지연로딩 : 객체가 실제 사용될 때 로딩
  - 즉시로딩 : JOIN SQL로 한번에 연관된 객체까지 미리 조회

- JPA 가장 중요한 2가지
  - 객체와 관계형 데이터베이스 매핑하기
  - 영속성 컨텍스트

- 영속성 컨텍스트

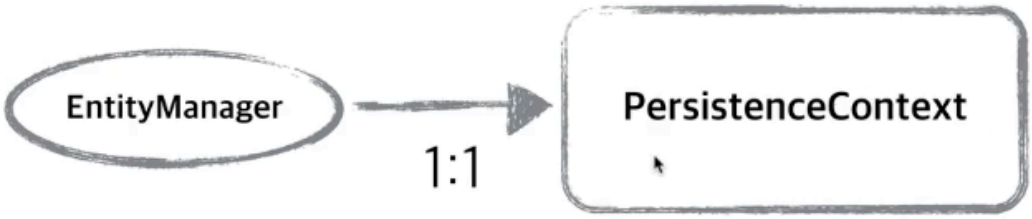
엔티티 매니저 팩토리와 엔티티 매니저



- 엔티티를 영구 저장하는 환경
- 영속성 컨텍스트는 논리적 개념
- 눈에 보이지 않는다.
- 엔티티 매니저를 통해 영속성 컨텍스트에 접근

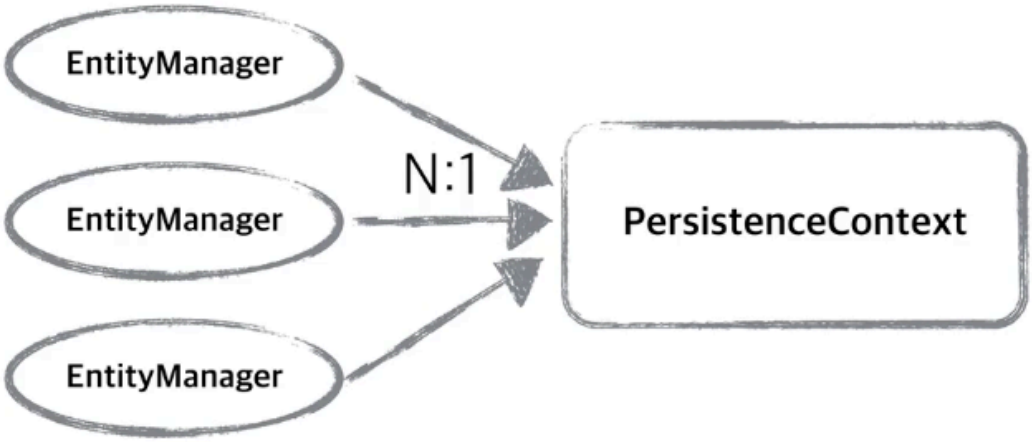
J2SE 환경

엔티티 매니저와 영속성 컨텍스트가 1:1

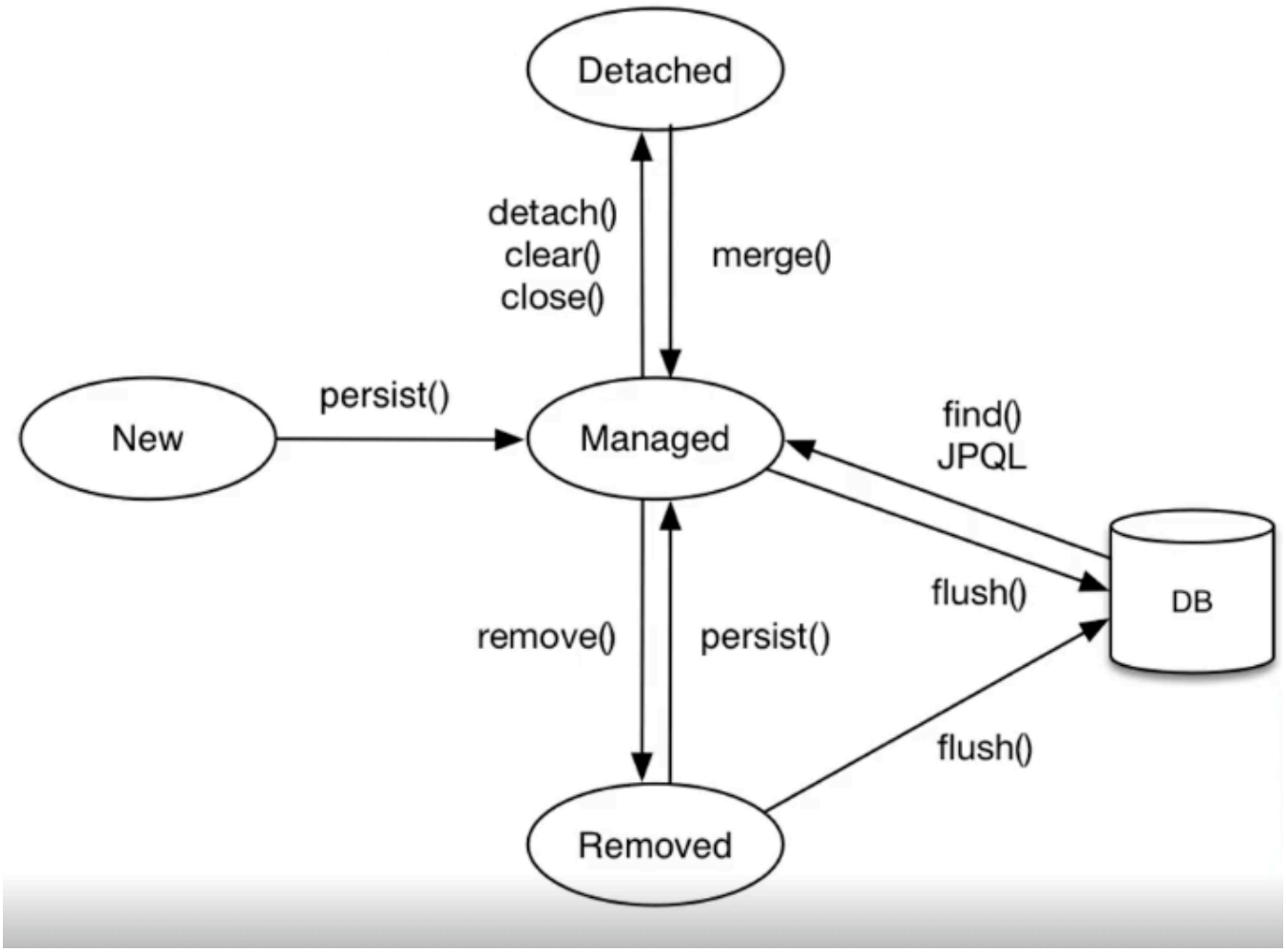


J2EE, 스프링 프레임워크 같은 컨테이너 환경

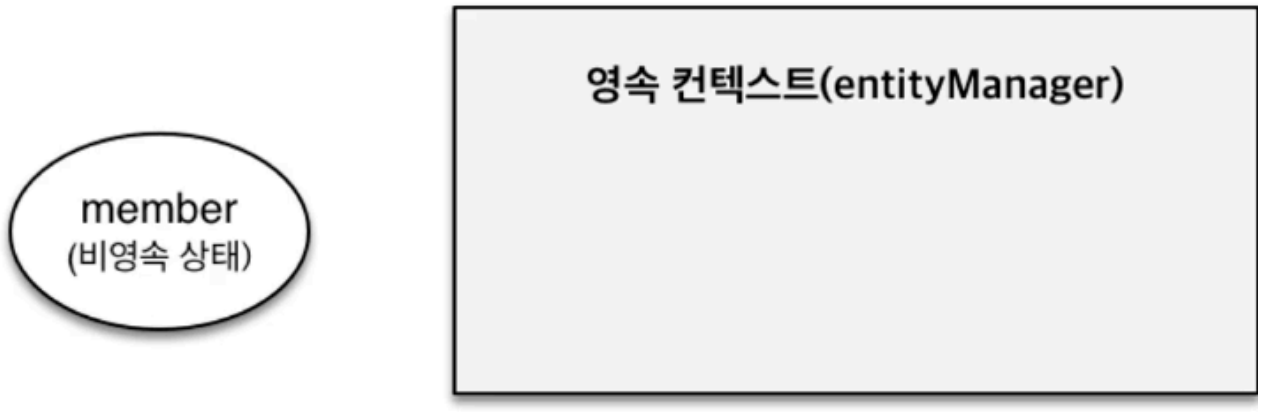
엔티티 매니저와 영속성 컨텍스트가 N:1



- 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


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

비영속



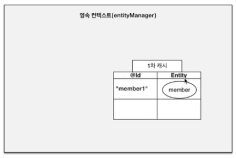

```
//객체를 생성한 상태(비영속)
Member member = new Member();
member.setId("member1");
member.setUsername("회원1");
```

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);
```

- 영속성 컨텍스트의 이점
  - 1차 캐시

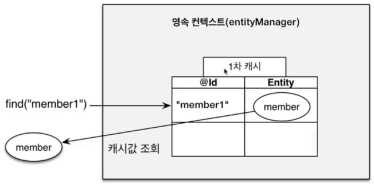
엔티티 조회, 1차 캐시



```
//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");
```



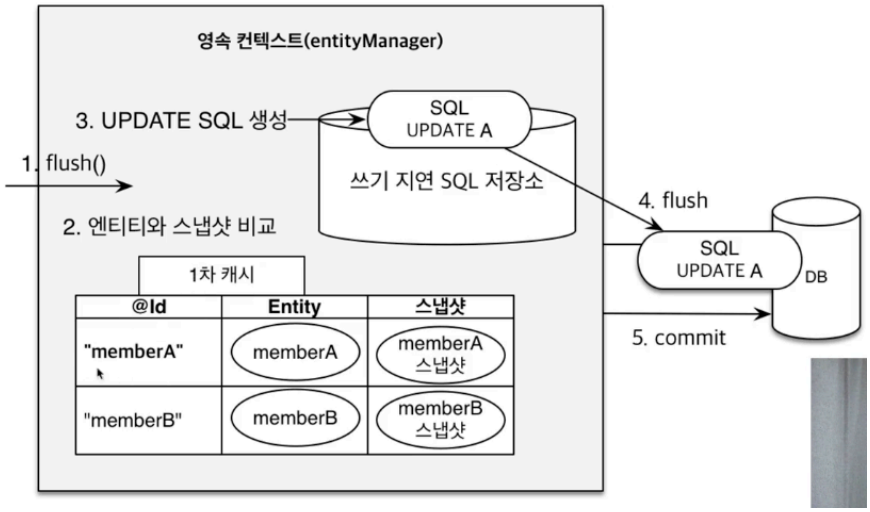
- 한 트랜잭션 안에서 이루어지기 때문에 큰 이점은 없다.

- 동일성 (identity) 보장
  - 1차 캐시로 반복 가능한 읽기 등급의 트랜잭션 격리 수준을 데이터베이스가 아닌 애플리케이션 차원 제공
  - 즉, 같은 entity 조회 결과 == 비교 시 true
- 트랜잭션을 지원하는 쓰기 지연 (Transactional write-behind)
  - 영속성 컨텍스트 ➜ 쓰기 지연 저장소에 저장 후
  - commit 시점에 flush & commit sql DB로 보낸다.
- 변경 감지 (Dirty Checking)
  - 영속 엔티티 조회 후 영속 엔티티 데이터 수정하면
  - 영속 컨텍스트

    [Application]] flush() ➜ 엔티티와 스냅샷 비교 ➜ update sql 생성 (쓰기지연 SQL저장소)

    [DB] ➜ flush ➜ SQL ➜ commit

변경 감지
(Dirty Checking)



- 지연로딩 (Lazy Loading)

- Flush : 영속성 컨텍스트의 변경내용을 데이터베이스에 반영
- Flush 발생
  - 변경감지
  - 수정된 엔티티의 쓰기 지연 SQL 저장소에 등록
  - 쓰기 지연 SQL 저장소의 쿼리를 데이터베이스에 전송 (등록 수정 삭제 쿼리)

- Detached
  - 특정 Entity만 상태 변경 가능하다.
  - Detached 되면 영속 컨택스트에서 사라진다.

#dev/jpa/tutorial