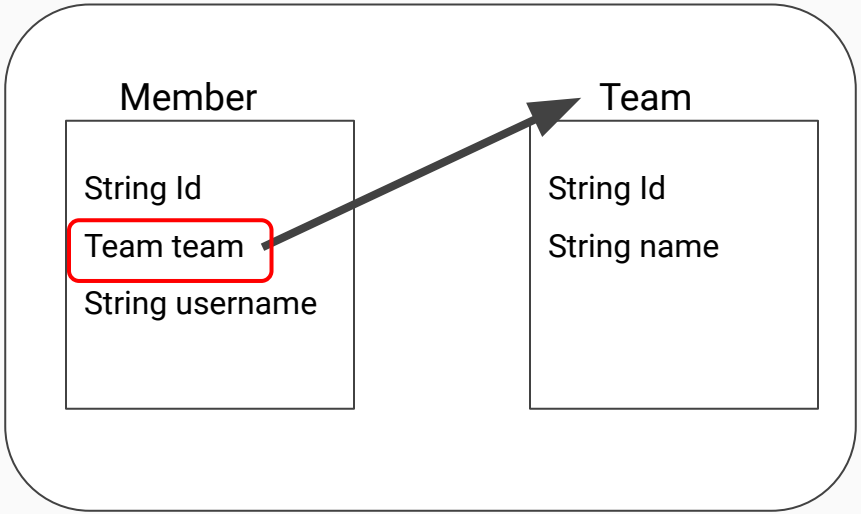
Chap08. 프록시와 연관관계 관리

-꼬막조림

1. 프록시

1. 프록시

Object

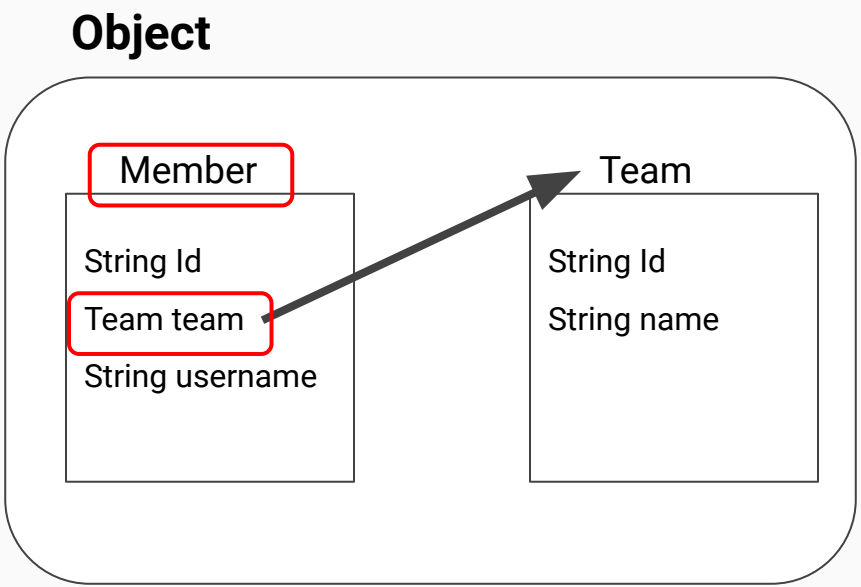



```
public Stirng printUser(String memberId) {
    Member member =
            em.find(Member.class, memberId);
    System.out.println(member.getUsername());
}
```

Member Entity를 조회하면서 Team Entity도 함께 조회함. 하지만 Team Entity는 사용되지 않음.

불필요한 Team Entity 조회로 자원 낭비

1. 프록시

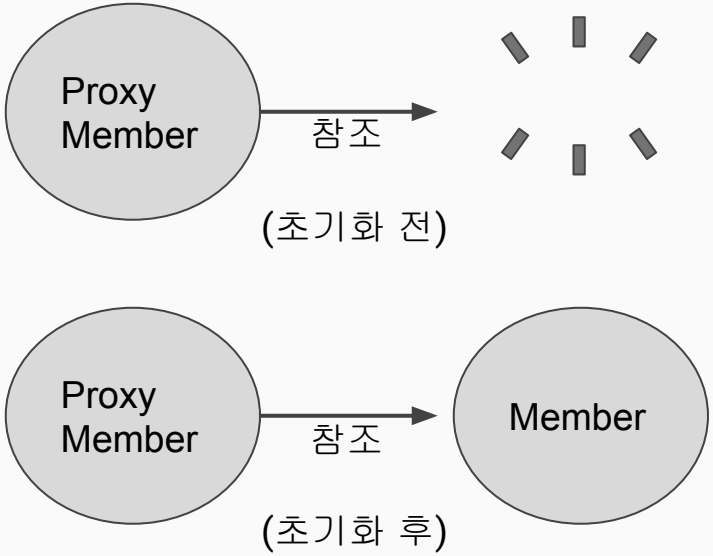


Member 객체에 속한 Team team Entity를 실제로 사용할때만 조회하는 기능을 **지연로딩**이라고 함.

Member클래스에 지연로딩을 사용 했을때 Member 클래스와 Team team 변수는 각각 가짜 Member객체와 Team 객체로 만들어지는데 이를 **프록시 객체** 라고 함.

※Proxy = 대리, 라는 뜻

1. 프록시



(지연 로딩&프록시 사용방법)
Member member =
em.getReference(Member.class, “id1”);

(이제 member객체는 proxy임.)

Member를 상속받은 Proxy 객체는 현재 참조하고 있는 실제 Member Entity가 없다.
Proxy객체가 실제 Entity를 참조하게 하는 과정을 **초기화** 라고 함.

1. 프록시

특징

- Proxy 객체는 처음 사용할 때 한 번만 초기화된다.
- Proxy 객체가 초기화 되면 실제 객체의 참조를 가지게 된다.
- Proxy 객체는 참조 객체의 타입과 다르다.(Member_$$_javassist_0)
- 실제 엔티티가 있다면 Proxy 객체를 사용하지 않고 실제 엔티티를 쓴다.
- 초기화는 영속상태에서만 가능하다.

2. 즉시 로딩과 지연 로딩

2. 즉시 로딩과 지연 로딩

즉시 로딩

- 엔티티를 조회할 때 연관된 엔티티도 함께 조회한다.
- @ManyToOne(fetch=FetchType.EAGER)
- Member member 클래스 조회시
  Team team 변수도 조회해야 하는데 각각 한번씩 두번의 Select 쿼리를 사용하지 않고 Join을 이용하여 한번의 쿼리로 처리한다.

지연 로딩

- 연관된 엔티티를 실제 사용할 때 조회한다.
- @ManyToOne(fetch=FetchType.LAZY)
- // team은 프록시 객체
  Team team = member.getTeam();
  // team 프록시 객체 초기화
  team.getName();

2. 즉시 로딩과 지연 로딩

JPA 기본 fetch 전략

- @Many**ToOne**, @One**ToOne** : 즉시 로딩(FetchType.EAGER)
- @One**ToMany**, @Many**ToMany** : 지연 로딩(FetchType.LAZY)

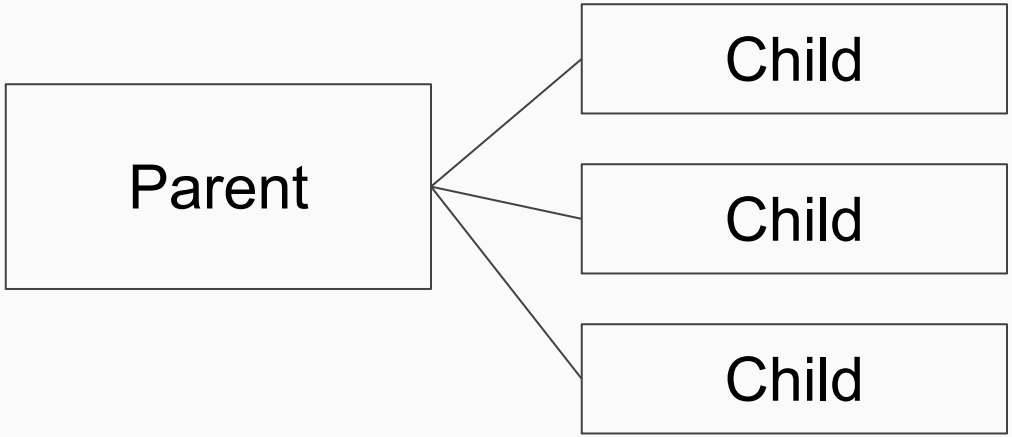
  List<Team>처럼 컬렉션을 로딩하는 것은 비용이 많이 들고 잘못하면 너무 많은 데이터를 로딩할 수 있기 때문.

  ToOne, ToMany 상관없이 모든 연관관계에 지연 로딩을 쓰고, 필요시 즉시로딩을 사용하도록 해주는게 좋음.

3. 영속성 전이: CASCADE

3. 영속성 전이: CASCADE

특정 엔티티를 영속 상태로 만들 때 연관된 엔티티도 함께 영속 상태로 만들고 싶으면 영속성 전이(transitive persistence) 기능을 사용.



영속성 전이를 사용하면 Parent 엔티티를 저장 할 때 Child 엔티티도 저장이 됨.

3. 영속성 전이: CASCADE

Parent

```
@Entity
public class Parent {

    @Id @GeneratedValue
    private Long id;

    @OneToMany(mappedBy = “parent”,
          cascade = CascadeType.PERSIST)
     private List<Child> children =
         New ArrayList<Child>();
     ...
}
```

Child

```
@Entity
public class Child {

    @Id @GeneratedValue
    private Long id;

    @ManyToOne
     private Parent parent;
     ...
}
```

3. 영속성 전이: CASCADE

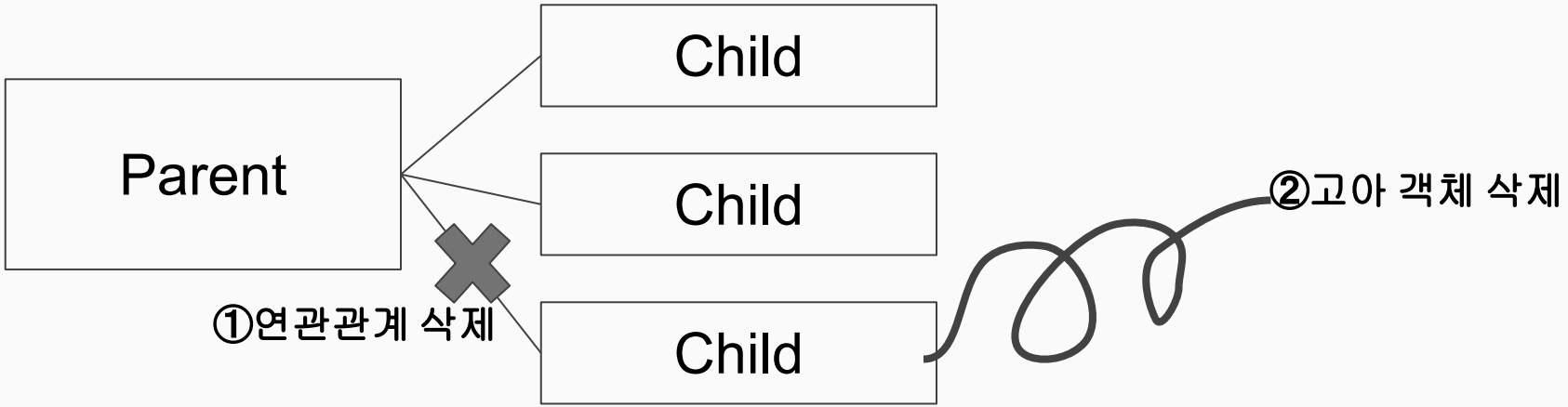
```
private static void saveWithCascade(EntityManager em) {

    Child child1 = new Child();
    Child child2 = new Child();

    Parent parent = new Parent();
    child1.setParent(parent);
    child2.setParent(parent);
    parent.getChildren().add(child1);
    parent.getChildren().add(child2);

    em.persist(parent); // child1, child2 도 함께 persist 됨.

}
```

```
private static void removeWithCascade
    (EntityManager em)
{

    Parent findParent
        = em.find(Parent.class, 1L);

    // child1, child2도 함께 remove
    em.remove(findParent);

}
```

4. 고아 객체

4. 고아 객체

부모 엔티티와 연관 관계가 끊어진 자식 엔티티를 고아 객체(Orphan obj)라고 하는데, JPA는 고아 객체를 자동으로 삭제 하는 옵션이 있다.



4. 고아 객체

Parent

```
@Entity
public class Parent {

    @Id @GeneratedValue
    private Long id;

    @OneToMany(mappedBy = “parent”,
          orphanRemoval = true)
     private List<Child> children =
         New ArrayList<Child>();
     ...
}
```

Child

```
@Entity
public class Child {

    @Id @GeneratedValue
    private Long id;

    @ManyToOne
     private Parent parent;
     ...
}
```

4. 고아 객체

```
private static void removeOrphanObj(EntityManager em) {
    Parent parent1 = em.find(Parent.class, 1L);
    parent1.getChildren().remove(0); // 특정 자식 제거
    parent1.getChildren().clear(); // 모든 자식 제거
}
```

4. 고아 객체

CascadeType.ALL 과 orphanRemoval = true 를 동시에 사용 하면 부모 객체만으로 자식의 생성과 삭제를 사용 할 수있음.

```
private static void casecadeAndOrphan(EntityManager em) {

    Parent parent1 = em.find(Parent.class, 1L);
    Child child1 = new Child();
    parent.addChild(child1);
    em.persist(parent1); // child1도 따라서 저장
    parent.getChildren().remove(child1); // 부모와 연관관계가 끊어지면 자식 객체 자동 삭제.

}
```

감사합니다.