영속성 전이와 고아객체

영속성 전이와 고아객체

```
@Entity
public class Parent {
  @Id
  @GeneratedValue
  private Long id;

  private String name;


  // OneToMany 에 CascadeType.ALL (영속성 전이)를 걸어주었다.
  // parent를 persist 할 때에 아래 리스트에 속해있는 것들을 모두 persist 할거야!
  @OneToMany(mappedBy="parent", cascade = CascadeType.ALL)
  private List<child> childList = new ArrayList<>();

  public void addChild(Child child) {
    childList.add(child);
    child.setParent(this);
  }
}
```

- 영속성 전이가 걸려있으면 em.persist(parent) 할 때에 연관되어있는 자식들까지 같이 persist 해주는 역할을 하게된다.
- **영속성 전이는 연관관계를 매핑하는 것과 전혀 관계가 없음**. (엔티티를 영속화할 때 연관된 엔티티도 함계 영속화하는 편리함을 제공할 뿐이다)

cascade의 종류!

- **ALL: 모두적용**
- **PERSIST: 영속(저장할때만)**
- **REMOVE: 삭제**
- MERGE: 병합
- REFRESH: 리프레시
- DETACH: DETACH

cascade를 쓰면 안 되는 경우

> child를 parent만 관리하는 경우에는 써도 되지만, 다른 entity에서도 child를 같이 관리하는 경우에는 cascade를 걸어서는 안된다. 운영이 너무 힘들어짐! 단일 엔티티에 종속적일 때만 쓸 것...(LifeCycle이 동일할 때)

고아객체

고아객체 제거 : 부모 엔티티와 연관관계가 끊어진 자식 엔티티를 자동으로 삭제

```
// @OneToMany annotation에 orphanRemoval = true 가 추가되었다.
@OneToMany(mappedBy="parent", cascade = CascadeType.ALL, orphanRemoval = true)
  private List<child> childList = new ArrayList<>();
```

고아객체 제거시 주의 사항!!

> 참조가 제거된 엔티티는 다른곳에서 참조하지 않는 고아객체로 보고 삭제하는 기능이다.
> orphanRemoval = true 로 사용하려면 참조하는 곳이 하나인 경우 / 특정 엔티티가 개인 소유일 경우에만 사용해야한다!! 또한 @OneToMany, @OneToOne에서만 사용가능하다.
>
> 개념적으로 부모를 제거하면 자식은 고아가 되는데, 고아 객체 제거 기능을 이용하면 부모 제거시 자식도 함께 제거가 되므로 마치 CascasdeType.REMOVE와 같이 동작한다.

즉시로딩 vs 지연로딩

```
@ManyToOne(fetch = FetchType.EAGER) // 즉시로딩
트랜잭션 시작시점에 쿼리를 모두 끌어와서 먼저 로딩

@ManyToOne(fetch = FetchType.LAZY) // 지연로딩
필요할 때 쿼리를 실행
```

주의점!

가급적 지연 로딩만 사용해야한다.

- 즉시 로딩을 적용하면 예상하지 못한 SQL이 발생할 수 있다.
- 즉시 로딩은 JPQL에서 N+1 문제를 일으킨다.
- @ManyToOne, @OneToOne은 기본이 즉시 로딩이기 때문에 LAZY로 설정해주어야 한다.

> N+1문제 : 처음 쿼리 하나를 날렸는데, 그것때문에 추가 쿼리 N개가 나가게 되는 문제점

FetchJoin (페치조인)

SQL의 조인 종류가 아니다. JPQL에서 성능 최적화를 위해서 제공하는 JPQL 전용 조인이다.

연관된 엔티티나 컬렉션을 SQL 한번에 함께 조회하는 기능! *(한방 쿼리)

즉시로딩과 비슷한 개념으로, 원하는 타이밍에 원하는 객체를 동적으로 조회가능

```
// JPQL
select m from Member m join fetch m.team

// SQL
SELECT M.*, T.* FROM MEMBER M INNER JOIN TEAM T ON M.TEAM_ID=T.ID
```

fetch join의 사용 예

```
String jpql = "select m from Member m join fetch m.team";
List<Member> members = em.createQuery(jpql, Member.class).getResultList();
```

```
for(Member member : members) {
  System.out.println("username = " + member.getUsername() + ", " +
                     "teamName = " + member.getTeam().name());
}
```

1:N JOIN을 하면 데이터 뻥튀기가 일어날 수 있다. 조심해야함

- distinct를 사용하여 중복 제거하는 방법
- +다른 방법 추후 설명