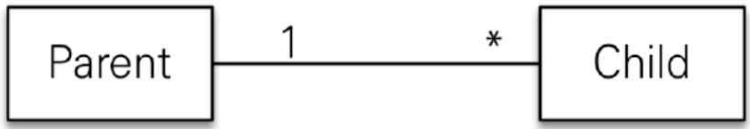
[JPA] 영속성 전이와 고아 객체

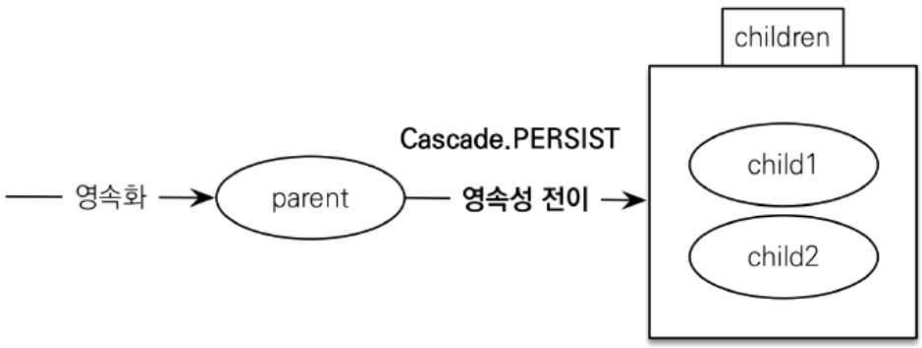
[JPA] 영속성 전이 : CASCADE

● 특정 엔티티를 영속 상태로 만들 때 연관된 엔티티도 함께 영속 상태로 만들고 싶을 때 사용하는 옵션
예) 부모 엔티티를 저장할 때 자식 엔티티도 함께 저장



- 부모 저장할 때, em.persist(parent)
- 연관된 컬렉션안의 자식 엔티티들 모두 다 함께 영속성 컨테스트에 저장하고 싶은 경우
- 영속성 전이 안 해주면 자식 개수만큼 em.persist(child) 일일이 해줘야함 ➡ 귀찮음

```
@OneToMany(mappedBy="parent", cascade = CascadeType.PERSIST)
```



주의

● 영속성 전이는 연관관계 매핑하는 것과 아무 관련 없음
● 엔티티를 영속화할 때 연관된 엔티티도 함께 영속화하는 편리함을 제공할 뿐이다!

CASCADE의 종류

- ALL : 모두 적용
- PERSIST : 영속
- REMOVE : 삭제
- MERGE : 병합
- REFRESH : REFRESH
- DETACH : DETACH

> 참조하는 곳이 한군데인 경우, 하나의 부모만 자식들을 관리하는 경우에만 의미있다.
> 게시판과 첨부파일과 같은 관계에서만 사용해야한다!
>
> 파일을 여러 엔티티에서 관리한다. 다른 게시물에서도 관리한다하면 사용하면 안된다!
> parent 하나만 단독으로 관리할 때만 사용해야한다 (소유자가 하나일 때)
>
> 단일 엔티티에 종속적인 경우에 사용! (라이프 사이클이 똑같기 때문)

1. 영속성 전이 설정 안할 경우

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Parent {
    @Id
    @GeneratedValue
    @Column(name = "parent_id")
    private Long id;

    @OneToMany(mappedBy = "parent")
    private List<Child> children = new ArrayList<Child>();
}

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Child {
    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name = "parent_id")
    private Parent parent;
}
```

만약 부모 1명에 자식 두명을 저장한다면?

```
    @Transactional
    public void saveNoCascade(){
        //부모 저장
        Parent parent = new Parent();
        em.persist(parent);

        //1번 자식 저장
        Child child1 = new Child();
        child1.setParent(parent);
        parent.getChildren().add(child1);
        em.persist(child1);

        //2번 자식 저장
        Child child2 = new Child();
        child2.setParent(parent);
        parent.getChildren().add(child2);
        em.persist(child2);
    }

    @Test
    @DisplayName("No Cascade")
    public void noCascadeTest(){
        relationTestService.saveNoCascade();
    }
```

2. 영속성 전이 설정할 경우

```
public class Parent {
    @Id
    @GeneratedValue
    @Column(name = "parent_id")
    private Long id;

    @OneToMany(mappedBy = "parent", cascade = CascadeType.PERSIST)
    private List<Child> children = new ArrayList<Child>();
}

    @Transactional
    public void saveCascade(){
```

```
        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);

        //부모저장, 연관된 자식들 저장
        em.persist(parent);
    }
```

3. 영속성 전이 삭제

방금 저장한 부모와 자식 엔티티를 모두 삭제하려면.. 다음과 같이 각각의 엔티티를 하나씩 제거...

영속성 전이는 엔티티 삭제할 때도 CascadeType.REMOVE로 설정하고, 부모 엔티티만 삭제하면 연관된 자식 엔티티도 함께 삭제하게 된다.

```
    @Transactional
    void cascadeDelete(){
        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);

        //부모저장, 연관된 자식들 저장
        em.persist(parent);
```

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

    @Test
    @DisplayName("CascadeDelete")
    public void cascadeDeleteTest(){
        relationTestService.cascadeDelete();
    }
```

[JPA] 고아 객체

> 부모 엔티티와 연관관계가 끊어진 자식 엔티티

● 고아 객체 제거 : 부모 엔티티와 연관관계가 끊어진 자식 엔티티를 자동으로 삭제

```
orphanRemoval = true
Parent parent1 = em.find(Parent.class, id);
parent1.getChildren().remove(0);
```

● 자식 엔티티를 컬렉션에서 제거함 ➡ 부모객체와의 연관관계 끊어짐

● DELETE FROM CHILD WHERE ID = ? ➡ DELETE 쿼리 발생한다.

주의

- 참조가 제거된 엔티티는 다른 곳에서 참조하지 않는 고아객체로 보고 삭제하는 기능
- 참조하는 곳이 하나일 때 사용해야한다!
- 특정 엔티티가 개인 소유할 때만 사용
- @OneToMany, @OneToOne만 가능 : 참조하는 쪽이 하나인 경우만 사용

> **개념적으로 부모를 제거하면 자식은 고아가된다.** 따라서 고아 객체 제거기능을 활성화하면, **부모를 제거할 때 자식도 함께 제거**된다. 이것은 **CascadeType.REMOVE처럼 동작**한다.

[JPA] 영속성 전이 + 고아객체, 생명주기

● CascadeType.ALL + orphanRemoval=true

● 스스로 생명주기를 관리하는 엔티티는 em.persist()로 영속화, em.remove()로 제거

  ○ JPA를 통해서 생명주기 관리

● 두 옵션을 모두 활성화하면 부모 엔티티를 통해서 자식 엔티티의 생명주기를 관리할 수 있다.

  ○ parent와 달리 child는 생명주기를 부모 엔티티가 관리

  ○ 굳이 DB로 따지면 DAO나 Repository가 없어도 된다는 의미

  ○ 생명주기를 부모가 관리하기 때문에

● 도메인 주도 설계(DDD)의 Aggregate Root 개념을 구현할 때 유용하다.