05 일대다 연관관계

일대다

- **일대다 연관관계**
  - 테이블에서는 일다대 연관관계를 외래키 하나로 관리
  - JPA에서는 일대다의 경우 @ManyToOne이나 @OneToMany로 구현할 수 있음
  - **@ManyToOne**은 child(post_comment)가 parent(post)의 PK를 가지는 자연스런 구조를 구현하도록 도움
  - 개발 편의성을 위해 JPA는 @OneToMany를 제공(객체관점에서 자주 사용)
    - 단방향 @OneToMany를 할 것인가? 양방향 @OneToMany를 할 것인가?

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

- "일"에 해당하는 PurchaseOrder와 "다"에 해당하는 Item 만들기

```java
@Entity
public class PurchaseOrder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PURCHASE_ORDER_ID")
    private Long id;
    private String userName;

    @OneToMany()//mappedBy = "order"
    private List<Item> items = new ArrayList<Item>();
```

```java
@Entity
public class Item {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ITEM_ID")
    private Long id;

    private String name;
    private int price;

    //@ManyToOne(fetch = FetchType.EAGER)
    //@JoinColumn(name = "PURCHASE_ORDER_ID")
    private PurchaseOrder order;
```

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

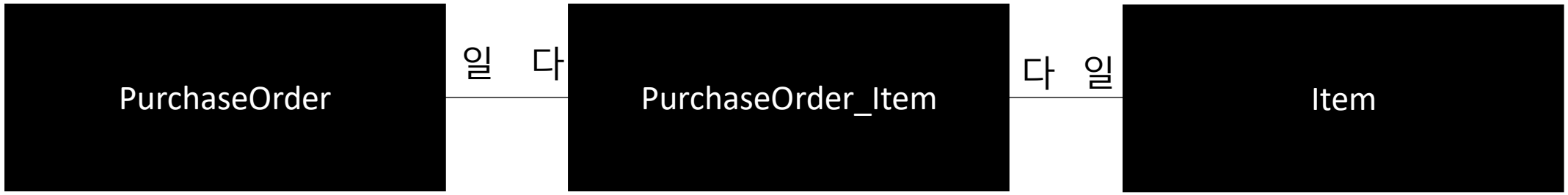
- **Main에 들어갈 테스트 코드**

```
Item item1 = new Item();
item1.setName("치킨");
Item item2 = new Item();
item2.setName("치즈볼");

PurchaseOrder order = new PurchaseOrder();
order.setUserName("kim");
order.getItems().add(item1);
order.getItems().add(item2);

em.persist(order);
em.persist(item1);
em.persist(item2);
em.flush();
em.clear();
```

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

- **일대다 단방향 연관관계 시 ERD**



  - DBA관점에서는 다대다 연관관계처럼 보임
  - 객체관점에서의 편이성을 위해 추가적인 테이블이 비효율적으로 생성

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

- **일대다 단방향 연관관계 with @JoinColumn**
  - 앞선 문제는 @JoinColumn을 명시하여 해결가능

```
@Entity
public class PurchaseOrder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PURCHASE_ORDER_ID")
    private Long id;
    private String userName;

    @OneToMany()
    @JoinColumn(name="PURCHASE_ORDER_ID")
    private List<Item> items = new ArrayList<Item>();
```

update 쿼리 실행 결과

- child는 FK가 설정되지 않은 채 일단 insert
- parent insert
- collection handling 단계에서 child의 FK가 없으므로 뒤 늦게 update쿼리 실행

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

- **일대다 단방향 연관관계 with @JoinColumn**
  - 위와 같은 맥락으로 remove에서도 추가적인 update 쿼리가 발생

```
order. getItems().remove(0);
```

```
update ……
```

  - ArrayList가 아닌 Set을 쓰면 달라질까? 달라지지 않음
- **대안은 일대다 양방향 연관관계**
  - 가장 적절한 방법은 일대일 연관관계에서 양방향을 걸어주는 것
  - 반대편에서 조회할 일이 없어도?
    - 운영 관점에서 이점이 있다면 양방향으로 연관관계 매핑하는 것은 큰 오버헤드가 아님

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대다

- **양방향 연관관계 주목할 점**
  - @ManyToOne연관관계는 LAZY로 설정
  - Parent인 Post는 addItem과 removeItem이라는 유틸리티(helper) 메소드를 정의해야 함. 양방향 연관관계의 동기화를 보장하는 안전장치
  - child인 Item는 equals와 hashCode 메소드를 오버라이딩 해야 함
    - 두 엔티티가 동일하다는 것은 RDB관점에서 PK가 같은지를 판별하는 것이지만 객체로 표현된 엔티티 관점에서는 인스턴스의 필드가 모두 같을 경우 같은 엔티티로 취급해야 함

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

@ManyToOne

- **@ManyToOne might be just enough**
  - 객체지향 관점에서 편리한 @OneToMany 어노테이션이 있지만 일대다 관계의 두 엔티티에 @OneToMany를 기본적으로 사용한다고 생각하면 안됨
  - child가 작은 상황에서만 사용하는 것이 바람직(@OneToMany보다 @OneToFew가 오히려 적절한 표현)
  - 따라서 "다(child)" 쪽에서 @ManyToOne 연관관계를 걸어주는 것이 바람직
  - 객체 관계에서 연관관계가 없지만 운영 측면에서 유리

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

Conclusion

- **정리**
  - 단방향 @OneToMany 보다는 양방향 @OneToMany 연관관계가 더 바람직
  - 반대편에서 @ManyToOne을 사용함으로써 **"다"쪽이 외래키를 관리**하는 자연스러운 구조가 됨
  - 객체관점에서 컬렉션 타입을 관리하는 @OneToMany를 사용하는 것이 편리해 보일지라도 @ManyToOne로 일대다를 관리하는 것이 좋음

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대일

- **일대일 연관관계**
  - Parent-Child관계로 표현
  - Parent는 비즈니스 적으로 중요하며 자주 참조되는 테이블
  - Child는 Parent와 일대일 연관관계를 맺고 있는 테이블
  - 책이 Parent, 원고가 Child가 될 수 있음
- **일대일 관계에서 FK의 위치**
  - 일대다, 다대일과 다르게 FK 위치를 Parent와 Child 중 하나에 지정할 수 있음
  - 각각의 장단점이 존재

[출처] https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

일대일

- **일대일 양방향(FK를 Parent에)**

```java
@Entity
public class Manuscript {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "MANUSCRIPT_ID")
    private Long id;

    private byte[] file;
    @OneToOne(mappedBy = "manuscript")
    private Book book;
```

```java
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "BOOK_ID")
    private Long id;
    private String ISBN;
    @OneToOne
    @JoinColumn(name = "MANUSCRIPT_ID")
    private Manuscript manuscript;
```

[출처] https://thorben-janssen.com/hibernate-tip-lazy-loading-one-to-one/

일대일

- 일대일 양방향(테스트)

```
Manuscript manuscript = new Manuscript();
Book book = new Book();
book.setISBN("ABCDEFG12");
book.setManuscript(manuscript);
manuscript.setBook(book);

em.persist(book);
em.persist(manuscript);
em.flush();
em.clear();
```

[출처] https://thorben-janssen.com/hibernate-tip-lazy-loading-one-to-one/

일대일

- **일대일 양방향(FK를 Child에)**

```java
@Entity
public class Manuscript {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "MANUSCRIPT_ID")
    private Long id;

    private byte[] file;
    @OneToOne
    @JoinColumn(name = "BOOK_ID")
    private Book book;
```

```java
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "BOOK_ID")
    private Long id;
    private String ISBN;

    @OneToOne(mappedBy = "book")
    private Manuscript manuscript;
```

[출처] https://thorben-janssen.com/hibernate-tip-lazy-loading-one-to-one/