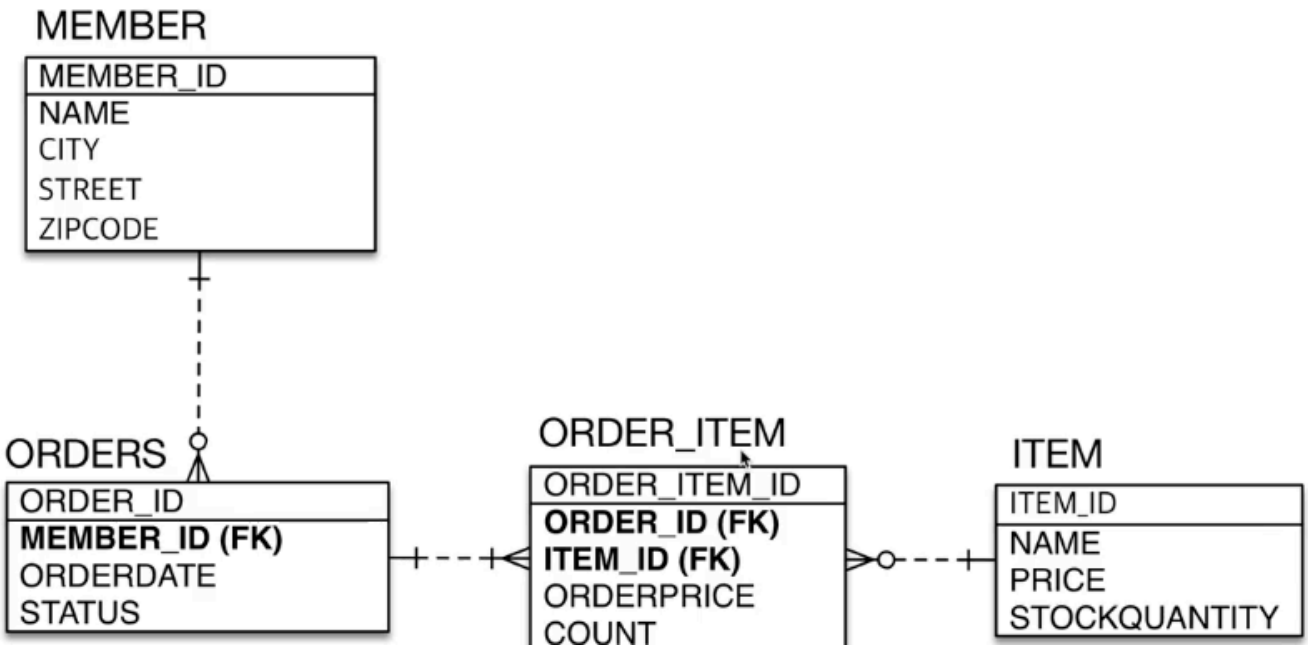
JPA 예제 및 연관관계 매핑 기초

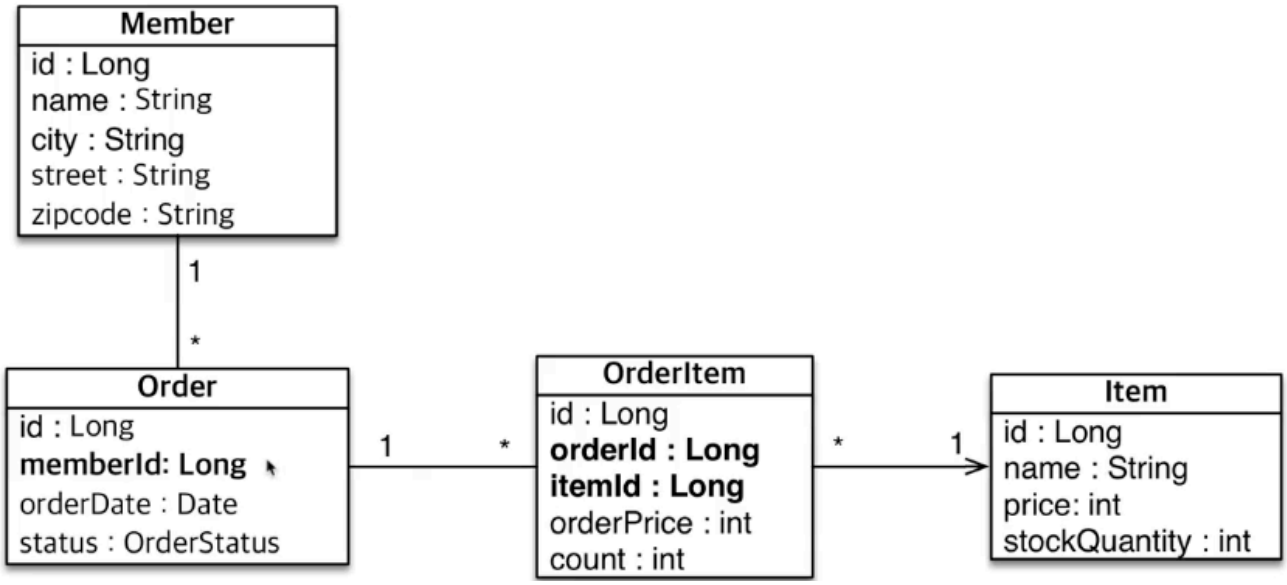
기능

- 회원기능
  - 회원등록
  - 회원조회
- 상품기능
  - 상품등록
  - 상품수정
  - 상품조회
- 주문기능
  - 상품주문
  - 주무내역조회
  - 주문취소

도메인 모델 분석

- 회원과 주문의 관계: 회원은 여러 번 주문할 수 있다 1:N
- 주문과 상품의 관계: 주문할 때 여러 상품을 선택할 수 있다. 상품도 여러 번 주문될 수 있다. 주문상품이라는 모델을 만들어 다대다 관계를 일대대, 다대일 관계로 풀어냄

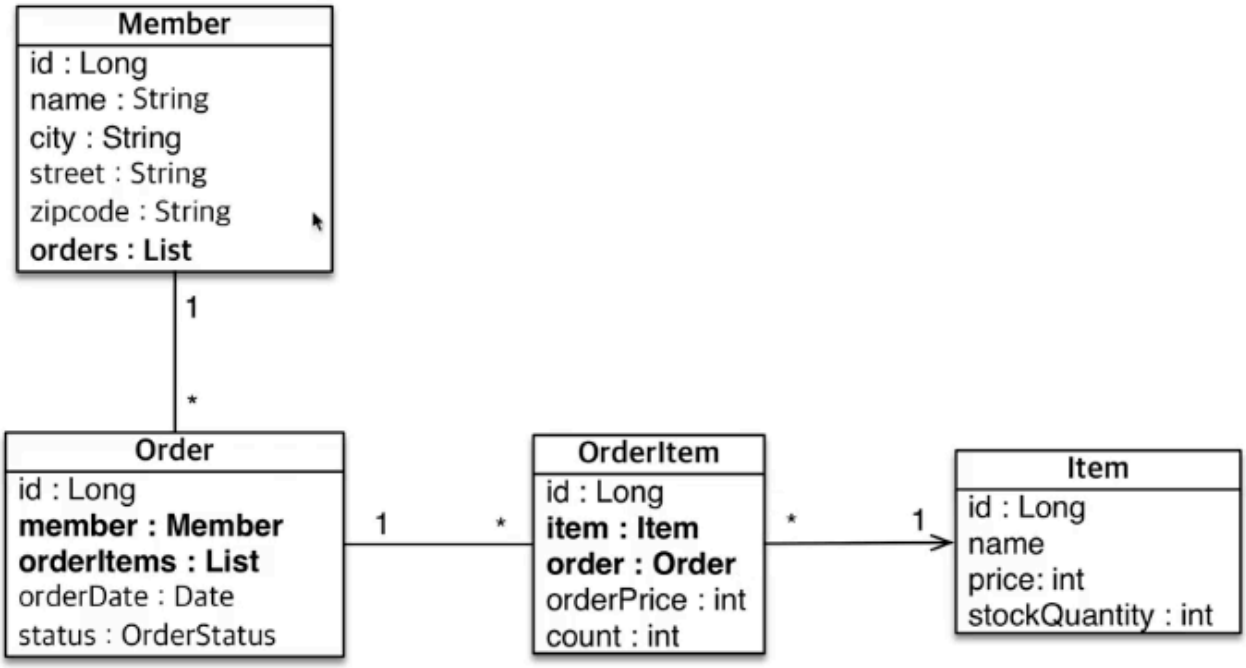




- 객체 설계를 테이블 설계에 맞춘 형식이다
  - 외래키 식별자를 직접 다룬다
  - 식별자로 다시 조회한다.
  - 즉 객체지향적인 방법이 아니다.
  - 협력관계를 만들기 어렵다.

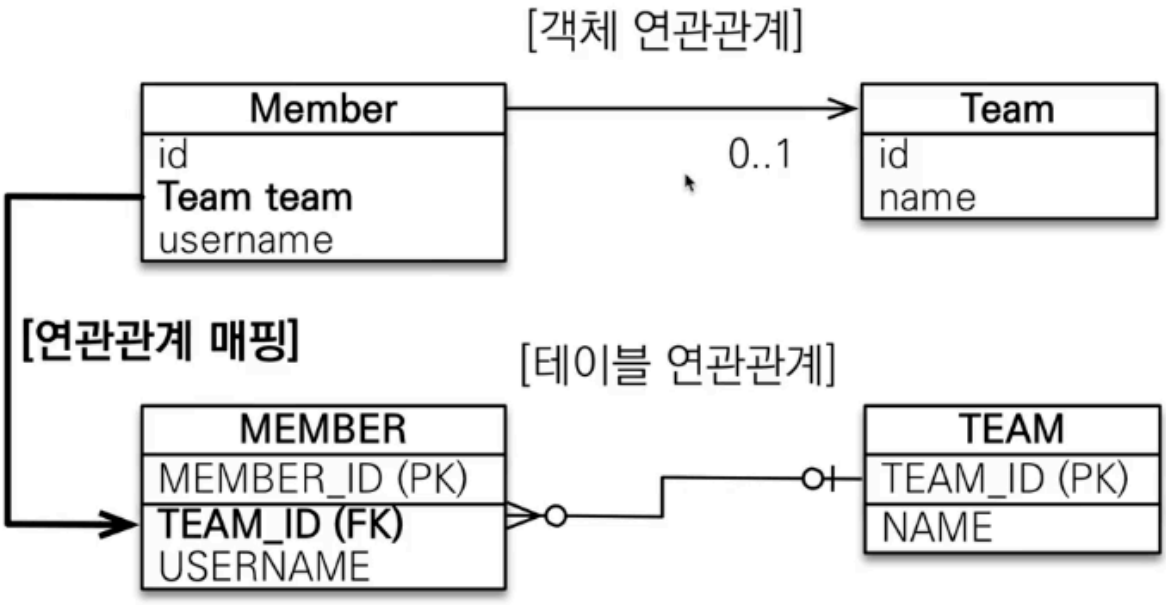
- 테이블의 외래키를 객체에 그대로 가져옴
- 객체 그래프 탐색이 불가능
- 참조가 없으므로 UML도 잘못됨

- 참조를 사용하도록 변경



- 연관관계 매핑 기초
  - 객체와 테이블 연관관계의 차이 이해
  - 객체의 참조와 테이블의 외래 키 매핑
  - 용어 이해
    - 방향(Direction): 단방향, 양방향
    - 다중성(Multiplicity): N:1, 1:N, 1:1, N:M
    - 연관관계의 주인 (Owner): 객체 양방향 연관관계는 관리 주인 필요

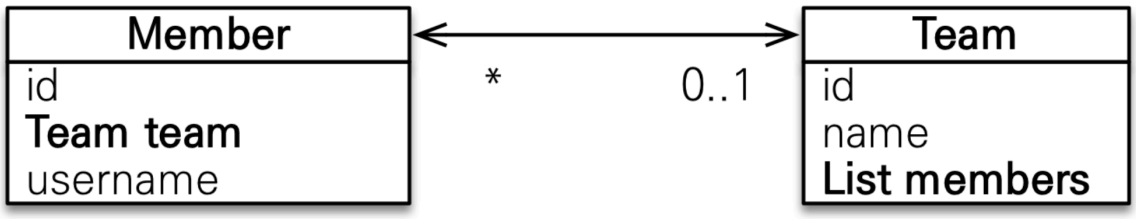
- 단반향
  - @ManyToOne
  - @JoinColumn(name=“TEAM_ID)”



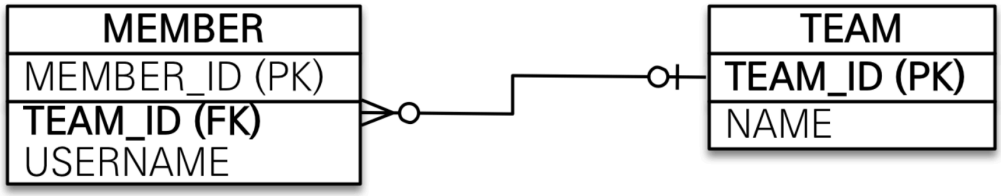

```
//팀 저장
Team team = new Team();
team.setName("TeamA");
em.persist(team);

//회원 저장
Member member = new Member();
member.setName("member1");
member.setTeam(team); //단방향 연관관계 설정, 참조 저장
em.persist(member);
```

- 양방향 연관관계와 연관관계의 주인

[양방향 객체 연관관계]



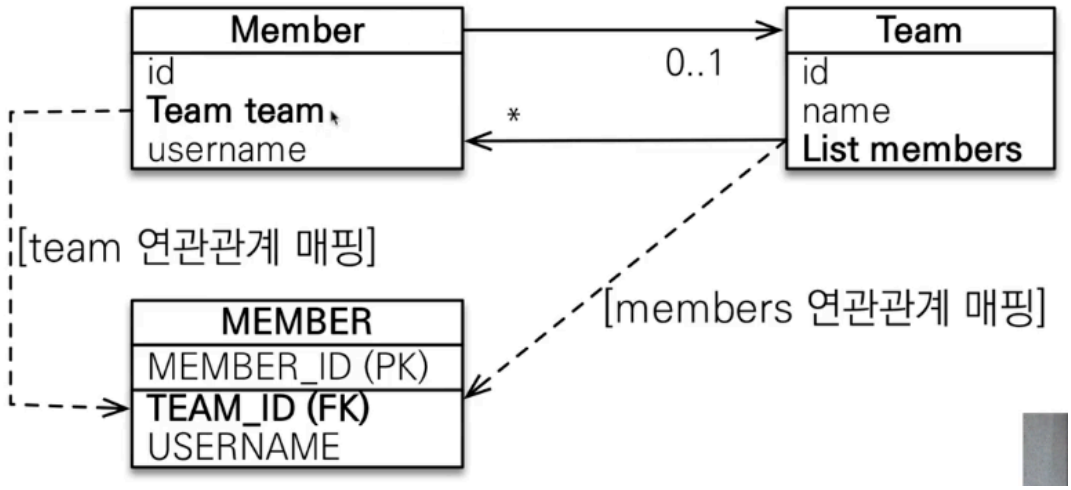
[테이블 연관관계]



-
- @OneToMany (mappedBy="team")
- List<Memeber> members = new ArrayList<Member>();

- mappedBy
  - 객체와 테이블간에 연관관계를 맺는 차이를 이해해야 한다.
    - 회원 ➔ 팀, 팀 ➔ 회원 객체는 연관관계 2개 (단방향 2개)

      테이블은 1개 (양방향)
  - 객체의 양방향 관계는 사실 양방향 관계가 아니라 서로 다른 단방향관계 2개이다.
  - 테이블은 외래키 하나로 두 테이블의 연관관계를 관리한다.



  -
  - 값을 바꿀 때 Member의 team의 값을 변경해야할지, Team의 값을 바꿔야할지 모른다.
  - 연관관계 주인
    - 객체의 두 관계 중 하나를 연관관계의 주인으로 지정해야한다.
    - 연관관계의 주인만이 외래 키를 관리 (등록, 수정)

        - 주인이 아닌 쪽은 읽기만 가능
        - 주인은 MappedBy 속성 사용하지 않는다.
        - 주인이 아니면 mappedBy 속성으로 주인 지정
    - 외래키가 있는 곳을 주인으로 정해라
    - 여기서는 Member.team 이 연관관계의 주인

- 주의해야할 사항
    - 연관관계의 주인에 값을 입력하지 않음
        - 양방향 매핑 시 연관관계의 주인에 값을 입력해야한다.

```
Team team = new Team();
team.setName("TeamA");
em.persist(team);

Member member = new Member();
member.setName("member1");

team.getMembers().add(member);
//연관관계의 주인에 값 설정
member.setTeam(team); //**

em.persist(member);
```

    - **순수 객체 상태를 고려해서 항상 양쪽에 값을 설정하자**
    - 연관관계 편의 메소드를 생성하자
        - 연관관계 객체가 있는지 확인하는 등등
        - 연관관계 한쪽에서만 하기
    - 양방향 매핑시에 무한 루프를 조심하자
        - 예: toString(), lombok, JSON 생성 라이브러리
- 단방향 매핑만으로도 이미 연관관계는 매핑 완료
- 양방향 매핑은 반대 방향으로 조회 (객체 그래프 탐색) 기능이 추가된 것 뿐이다.
- JPQL에서 역방향으로 탐색할 일이 많다.
- 단반향 매핑을 잘하고 양방향은 필요할 때 추가해도 된다.

- 가급적 단방향이 좋다 ➜ 단방향으로 개발하다가 양방향으로 데이터를 가져와야할 때 그때 양방향으로 추가해줘

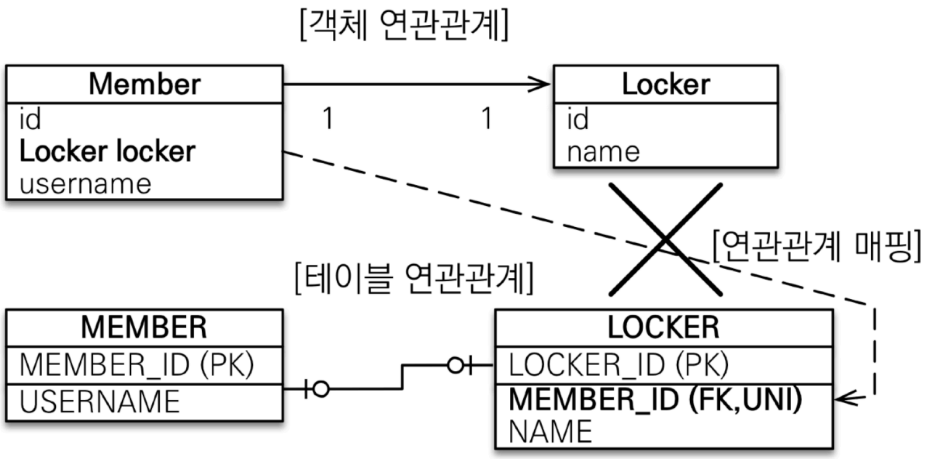
도 된다.

- 위의 예제에서 Member 가 Orders를 연관관계 양방향으로 가지고 있는게 좋은 것일까?
  관심사를 끊어내야할 건 끊어야한다. 사실 Member Orders를 가지고 있을 필요 없다.

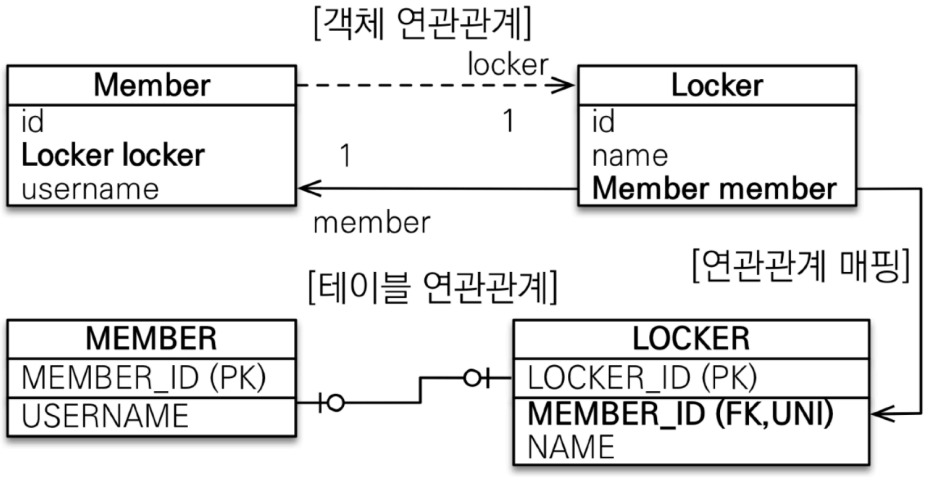
연관관계 매핑시 고려사항 3가지

- 다중성
  - 다대일 (실무에서 사용하면 안된다.)
  - N:1 | @ManyToOne
    - 양방향
      - 외래 키가 있는 쪽이 연관관계의 주인
      - 양쪽 서로 참조하도록 개발
  - 1:N | @OneToMany
    - 단방향 (지양하는 편)
      - 연관관계를 위해 추가로 Update SQL 실행 insert 이후 update 를 또 해줘야한다.
      - 실무에서 운영이 어려워진다.
      - 그래서 다대일 양방향으로 처리하는 경우가 많다.
    - 데이블 일대다 관계는 항상 다(N) 쪽에 외래 키가 있다.
    - 객체와 테이블의 차이 때문에 반대편 테이블의 외래키를 관리하는 특이한 구조
    - JoinColumn을 꼭 사용해야함. 그러지 않으면 조인테이블 방식을 사용
    - 양방향
      - 이런 매핑은 공식적으로 존재X
      - @JoinColumn(**insertable=false, updatable=false**)
      - **읽기 전용 필드**를 사용해서 양방향 처럼 사용하는 방법
      - **다대일 양방향을 사용하자**
  - 1:1 | @OneToOne
    - 일대일 관계는 그 반대도 일대일
    - 주 테이블이나 대상 테이블 중에 외래키 선택가능
      - 주 테이블 외래키
      - 대상테이블 외래키
    - 외래키에 데이터베이스 유니크 제약조건 추가
    - 다대일(@ManyToOne) 단방향 매핑과 유사
    - 다대일 양방향 매핑 처럼 **외래 키가 있는 곳이 연관관계의 주인**
    - 반대편은 mappedBy 적용
    - 주 테이블에 외래 키 단방향이 개발에는 편하다.

일대일: 대상 테이블에 외래 키 단방향



일대일: 대상 테이블에 외래 키 양방향



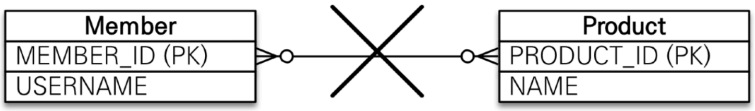
- **주 테이블에 외래 키**
  - 주 객체가 대상 객체의 참조를 가지는 것 처럼 주 테이블에 외래 키를 두고 대상 테이블을 찾음
  - 객체지향 개발자 선호
  - JPA 매핑 편리
  - 장점: 주 테이블만 조회해도 대상 테이블에 데이터가 있는지 확인 가능
  - 단점: 값이 없으면 외래 키에 null 허용
- **대상 테이블에 외래 키**
  - 대상 테이블에 외래 키가 존재
  - 전통적인 데이터베이스 개발자 선호

- 장점: 주 테이블과 대상 테이블을 일대일에서 일대다 관계로 변경할 때 테이블 구조 유지
- 단점: 프록시 기능의 한계로 **지연 로딩으로 설정해도 항상 즉시 로딩됨**(프록시는 뒤에서 설명)

- M:N | @ManyToMany

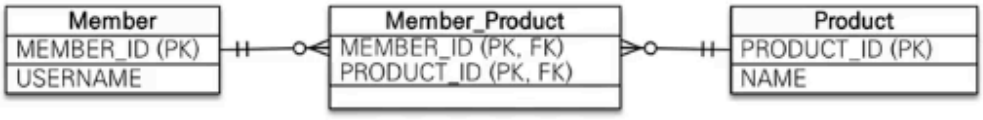
다대다

- 관계형 데이터베이스는 정규화된 테이블 2개로 다대다 관계를 표현할 수 없음
- 연결 테이블을 추가해서 일대다, 다대일 관계로 풀어내야함



다대다

- 객체는 컬렉션을 사용해서 객체 2개로 다대다 관계 가능

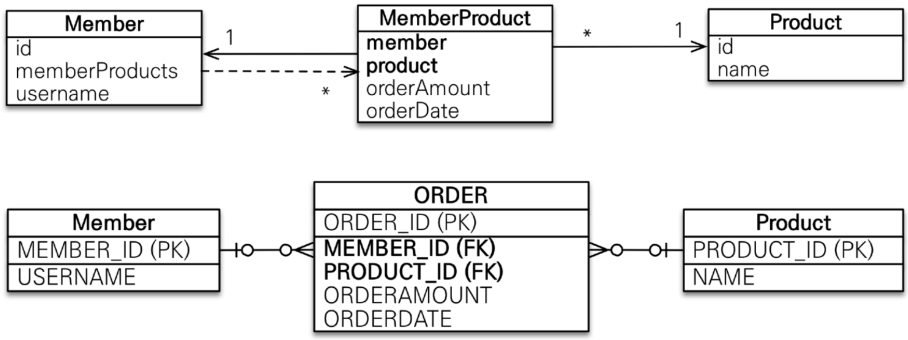


- **@ManyToMany** 사용
- **@JoinTable**로 연결 테이블 지정
- 다대다 매핑: 단방향, 양방향 가능
- **편리해 보이지만 실무에서 사용 하지 않는다**

- 연결 테이블이 단순히 연결만 하고 끝나지 않음
- 주문시간, 수량 같은 데이터가 들어올 수 있음

다대다 한계 극복

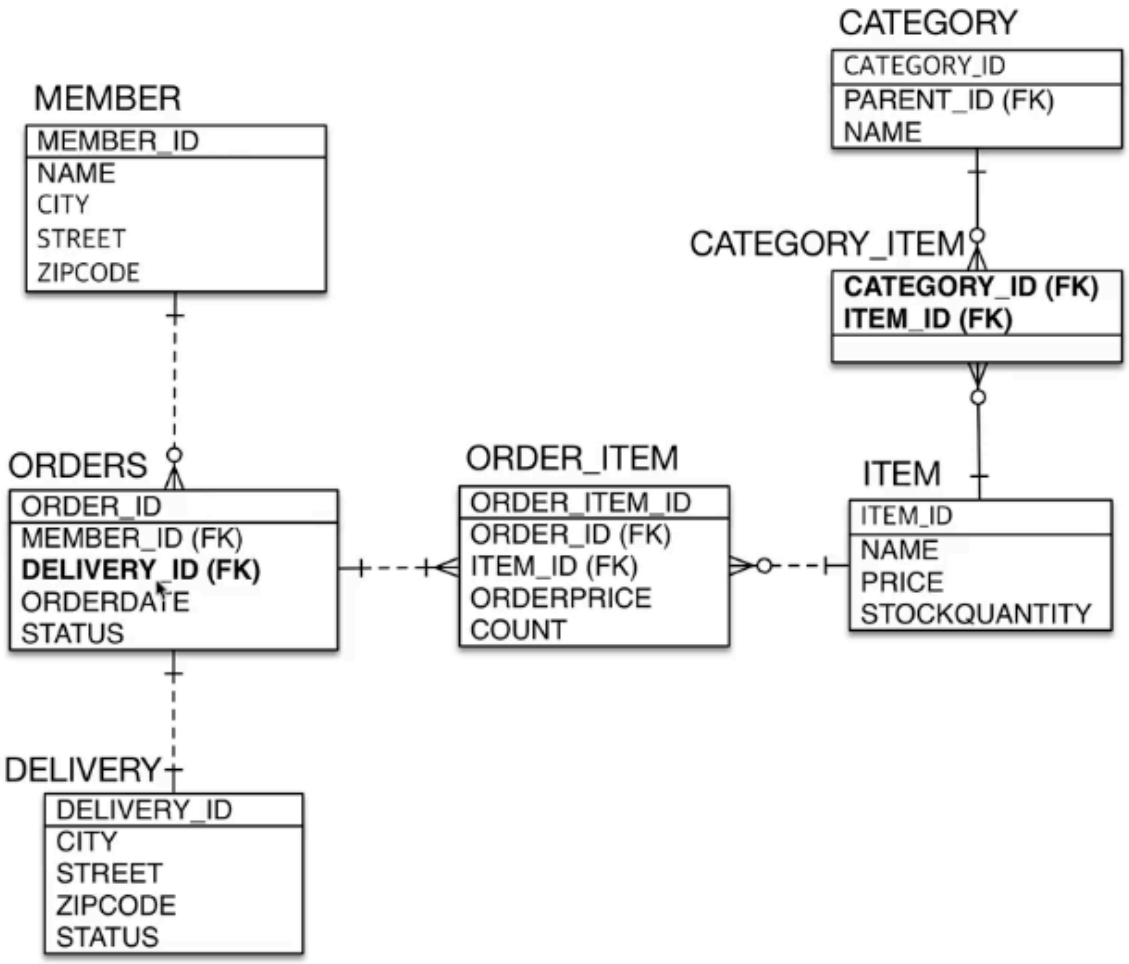
- **연결 테이블용 엔티티 추가(연결 테이블을 엔티티로 승격)**
- **@ManyToMany -> @OneToMany, @ManyToOne**



- 단방향, 양방향
  - 테이블
    - 외래 키 하나로 양쪽 조인 가능
    - 사실 방향이라는 개념이 없음
  - 객체
    - 참조용 필드가 있는 쪽으로만 참조가능
    - 한쪽만 참조하면 단반향
    - 양쪽이 서로 참조하면 양방향
- 연관관계의 주인
  - 테이블은 외래키 하나로 두 테이블이 연관관계를 맞음
  - 객체는 양방향 관계는 참조가 2군데
  - 연관관계 주인, 외래키를 관리하는 참조
  - 주인의 반대는 단순히 조회만

예시) ManyToMany를 예시를 위해 연관관계 예시

배송, 카테고리 추가 - ERD



- @JoinColumn

- 외래 키를 매핑할 때 사용

속성	설명	기본값
name	매핑할 외래 키 이름	필드명 + _ + 참조하는 테이블의 기본 키 컬럼명
referencedColumnName	외래 키가 참조하는 대상 테이블의 컬럼명	참조하는 테이블의 기본 키 컬럼명
foreignKey(DDL)	외래 키 제약조건을 직접 지정할 수 있다. 이 속성은 테이블을 생성할 때만 사용한다.	
unique nullable insertable updatable columnDefinition table	@Column의 속성과 같다.	

- @ManyToOne - 주요 속성

@ManyToOne - 주요 속성

- 다대일 관계 매핑

속성	설명	기본값
optional	false로 설정하면 연관된 엔티티가 항상 있어야 한다.	TRUE
fetch	글로벌 페치 전략을 설정한다.	- @ManyToOne=FetchType.EAGER - @OneToMany=FetchType.LAZY
cascade	영속성 전이 기능을 사용한다.	
targetEntity	연관된 엔티티의 타입 정보를 설정한다. 이 기능은 거의 사용하지 않는다. 컬렉션을 사용해도 제네릭으로 타입 정보를 알 수 있다.	

- @ OneToMany - 주요 속성

@OneToMany - 주요 속성

- 일대다 관계 매핑

속성	설명	기본값
mappedBy	연관관계의 주인 필드를 선택한다.	
fetch	글로벌 페치 전략을 설정한다.	- @ManyToOne=FetchType.EAGER - @OneToMany=FetchType.LAZY
cascade	영속성 전이 기능을 사용한다.	
targetEntity	연관된 엔티티의 타입 정보를 설정한다. 이 기능은 거의 사용하지 않는다. 컬렉션을 사용해도 제네릭으로 타입 정보를 알 수 있다.	

#dev/jpa/tutorial