9. 논리적 모델링 - 실습

#0.강의/2.데이터베이스로드맵/3.설계1

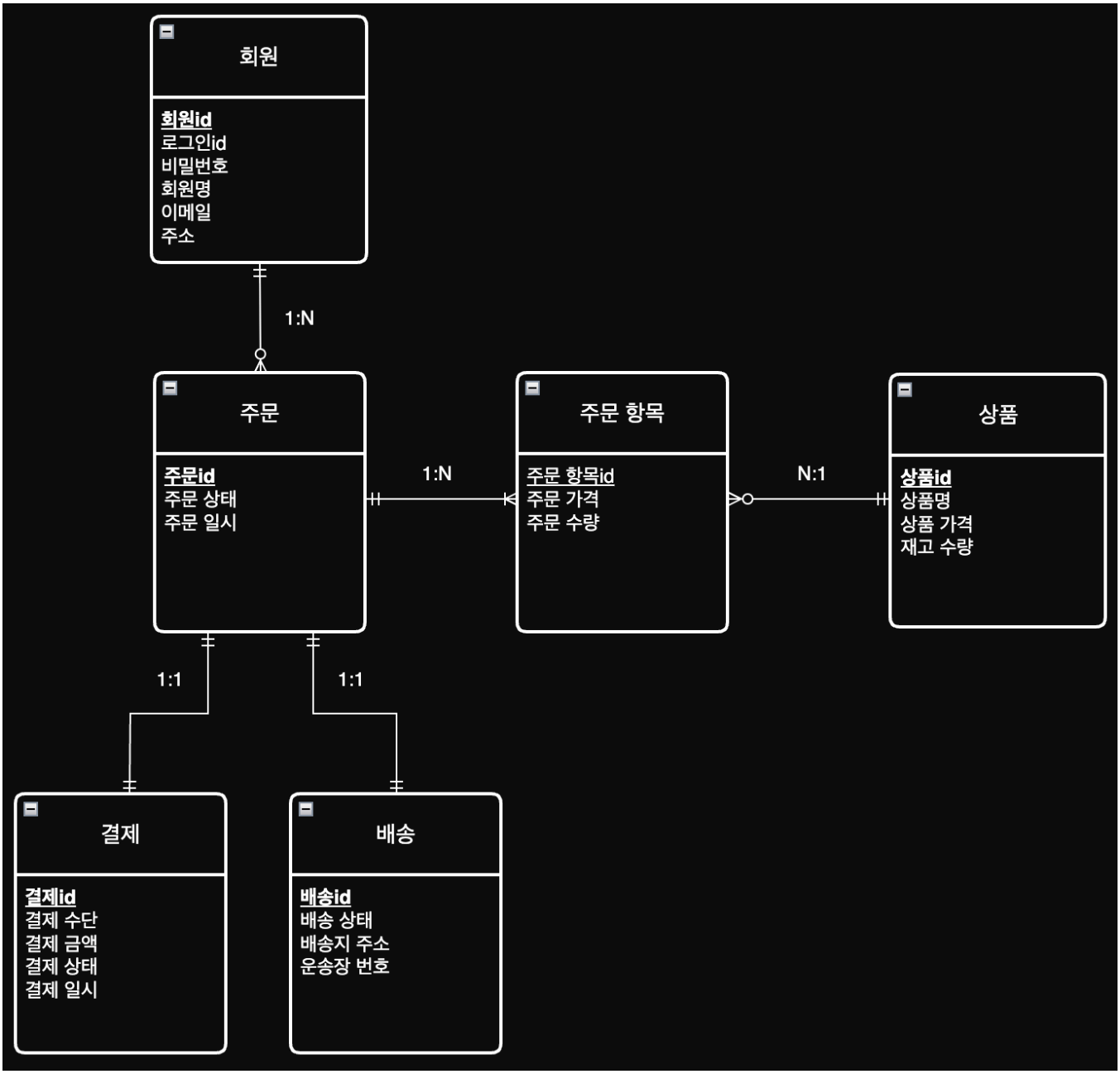
- /실전 논리적 모델링 - 시작
- /실전 논리적 모델링 - ERD 작성

실전 논리적 모델링 - 시작

개념적 모델링을 통해 비즈니스 요구사항을 ERD라는 청사진으로 만들었고, 논리적 모델링의 핵심 이론인 키(Key)와 관계(Relationship)에 대해서도 깊이 있게 학습했다. 이제 이 모든 지식을 하나로 합쳐, 우리의 '쇼핑몰' ERD를 실제 관계형 데이터베이스에 한 걸음 더 가까운 논리적 모델로 완성시킬 차례다.

우리가 만든 개념적 모델을 다시 한번 살펴보자.

[쇼핑몰 MVP 개념적 모델 ERD]



- **엔티티**: 회원, 상품, 주문, 결제, 배송, 주문 항목
- **관계**: 회원-주문(1:N), 주문-결제(1:1), 주문-배송(1:1), 주문-상품(M:N)은 주문 항목을 통해 1:N 관계로 해소

논리적 모델링은 이 개념적 모델을 관계형 데이터베이스의 언어(테이블, 컬럼, 키)로 번역하는 과정이다.
이 단계는 단순히 1:1로 번역하는 것에 그치지 않는다. 우리가 논리적 모델링 단계에서 학습한 수많은 트레이드오프들을 고민하면서 제품의 현실적인 제약과 개발 효율성, 성능까지 고려하여 모델을 다듬고 최적화하는 실용적인 판단이 필요하다.

개념에서 논리로: 변환의 법칙

논리적 모델링을 시작하기 전에, 개념적 모델의 추상적인 요소들이 논리적 모델의 어떤 구체적인 요소로 변환되는지 명

확히 이해해야 한다. 이 변환에는 다음과 같은 명확한 법칙이 존재한다.

1. 엔티티(Entity)는 테이블(Table)로 변환된다.

개념적 모델에서 정의한 데이터의 묶음, 즉 **엔티티**는 논리적 모델에서 데이터를 실제로 저장하는 구조인 **테이블**이 된다.

- `회원` 엔티티 → `회원` 테이블
- `상품` 엔티티 → `상품` 테이블
- `주문` 엔티티 → `주문` 테이블

2. 속성(Attribute)은 컬럼(Column)으로 변환된다.

각 엔티티가 가지고 있던 세부 정보, 즉 **속성**은 테이블의 각 **컬럼**으로 변환된다. 이 과정에서 우리는 각 컬럼에 저장될 데이터의 종류와 크기를 정의하는 **데이터 타입**(`VARCHAR`, `BIGINT`, `DATETIME` 등)을 결정하게 된다. 그리고 추가로 제약조건도 정한다.

- `회원` 엔티티의 `회원명` 속성 → `회원` 테이블의 `회원명 VARCHAR(50) NOT NULL` 컬럼
- `상품` 엔티티의 `가격` 속성 → `상품` 테이블의 `가격 INT` 컬럼

3. 식별자(Identifier)는 기본 키(Primary Key)로 변환된다.

각 엔티티를 유일하게 구분해주던 **식별자**는 테이블의 각 행(row)을 유일하게 보장하는 **기본 키(Primary Key)**가 된다.

여기서 우리는 앞서 배운 중요한 설계 원칙을 적용한다. 개념적 모델의 `회원id`와 같은 식별자는 비즈니스와 무관한 **대리 키(Surrogate Key)**를 사용하여 PK로 구현한다.

- `회원` 엔티티의 식별자 `회원id` → `회원` 테이블의 `회원id BIGINT NOT NULL AUTO_INCREMENT` 기본 키

4. 관계(Relationship)는 외래 키(Foreign Key)와 제약조건으로 변환된다.

개념적 모델에서 엔티티 사이를 연결하던 선, 즉 **관계**는 논리적 모델에서 테이블들을 연결하는 **외래 키(Foreign Key)**를 통해 구체화된다. 관계의 종류(카디널리티)에 따라 변환 방식이 달라진다.

- **일대다(1:N) 관계**: **'다(N)' 쪽 테이블**이 '일(1)' 쪽 테이블의 기본 키를 외래 키로 가진다.
  - `회원(1)`과 `주문(N)`의 관계 → `주문` 테이블에 `회원id` 외래 키 컬럼이 추가되어 `회원` 테이블의 `회원id`를 참조한다.

- **일대일(1:1) 관계**: 두 테이블 중 하나가 상대방의 기본 키를 외래 키로 가지며, 해당 외래 키에는 **UNIQUE 제약조건**이 추가되어야 한다. 주로 더 부가적이거나, 나중에 생성되는 '자식' 역할의 테이블에 외래 키를 둔다.
  - `주문(1)`과 `배송(1)`의 관계 → `배송` 테이블에 `주문ID` 외래 키 컬럼을 추가하고, 여기에 `UNIQUE` 제약조건을 설정한다.

- **다대다(M:N) 관계**: 두 엔티티 사이의 **관계 자체가 새로운 테이블**로 변환된다. 이 테이블을 **연결 테이블**

(Junction Table)이라 부른다. 참고로 개념적 모델에서 양쪽으로 뻗어 나가던 다대다 관계는 관계형 데이터베이스에서 직접 표현할 수 없다. 그래서 중간에 연결 테이블이라는 새로운 테이블을 만들어 두 개의 일대다(1:N) 관계로 풀어낸다.

- `주문(M)`과 `상품(N)`의 관계 → `주문 항목`이라는 새로운 연결 테이블이 생성된다. 이 테이블은 `주문` 테이블을 참조하는 `주문ID` 외래 키와 `상품` 테이블을 참조하는 `상품ID` 외래 키를 모두 가진다.
- 우리의 경우 이미 자신의 속성을 가지는 `주문 항목`이라는 연관 엔티티를 개념적 모델링 단계에서 찾았다. 이 경우 연관 엔티티를 연결 테이블로 만들면 된다.

요약: 개념적 모델 vs 논리적 모델

개념적 모델 요소	→	논리적 모델 요소 (관계형 데이터베이스)	예시
엔티티 (Entity)	→	**테이블** (Table)	`회원` → `회원` 테이블
속성 (Attribute)	→	**컬럼** (Column) + 데이터 타입	회원명 → 회원명 `VARCHAR(50)`
식별자 (Identifier)	→	**기본 키** (Primary Key)	`회원id` → `회원id` (PK)
관계 (Relationship)	→	**외래 키** (Foreign Key) + 제약조건	회원-주문 관계 → 주문.`회원id` (FK)
다대다 관계 (M:N)	→	**연결 테이블** (Junction Table)	`주문-상품` 관계 → `주문 항목` 테이블

이번 실전 논리적 모델링의 목표는 다음과 같다.

- 우리가 합의한 현대적 설계 원칙을 적용한다: **대리 키-PK, 자연 키-UNIQUE, 관계-비식별**
- 모든 테이블의 기본 키(PK)와 외래 키(FK)를 명확하게 정의한다.
- 개념적 모델의 엔티티는 테이블로, 속성은 컬럼으로 변환한다.

이 규칙들을 머릿속에 그리고, 우리가 만들었던 쇼핑몰 개념적 모델을 하나씩 변환해보자.

참고: 논리적 모델링과 물리적 모델링, 무엇이 다를까?

관계형 데이터베이스 설계는 개념적 설계 이후에 '논리적 모델링'과 '물리적 모델링' 두 단계로 나뉘지만, 실제로는 그 경계가 모호할 때가 많다.

관계형 데이터베이스별 물리적 타입 비교

똑같은 '자동 증가 숫자'라도, 실제 관계형 데이터베이스마다 구현하는 방식(데이터 타입)이 다음과 같이 다르다.

논리적 타입	MySQL	Oracle	PostgreSQL
자동 증가하는 큰 숫자	`BIGINT AUTO_INCREMENT`	`NUMBER` (+ `SEQUENCE`)	`BIGSERIAL`
가변 길이 문자열	`VARCHAR(n)`	`VARCHAR2(n)`	`VARCHAR(n)`

논리적 모델링: 데이터의 청사진 그리기
논리적 모델링은 이미 관계형 데이터베이스를 사용하기로 결정한 단계이다. 그리고 어떤 관계형 데이터베이스를 사용할지 정하기 전에, 데이터의 **구조와 관계**를 정의하는 단계이다. 정리하면 관계형 데이터베이스 모델에 맞추어 일종의 데이터 청사진을 그리는 과정이다.

- **핵심**: 관계형 데이터베이스 이론에 맞추어 데이터의 논리적 구조를 설계한다. (MySQL, Oracle 등 특정 기술과 무관)
- **예시**: "회원에게는 '회원명' 속성이 있고, **문자열(String)** 형태다." 라고 정의하는 것이다. 이렇게 하면 특정 기술과 무관하게 정할 수 있다.

물리적 모델링: 실제 창고에 물건 쌓기
물리적 모델링은 논리적 모델을 바탕으로, **실제 데이터베이스에 어떻게 데이터를 저장할지** 구체적으로 정하는 단계이다.

- **핵심**: 특정 데이터베이스(예: MySQL)에 최적화된 형태로 구현한다.
- **예시**: 논리 모델의 '문자열'을 MySQL에서 `VARCHAR(50)`으로 할지, `TEXT`로 할지 성능과 효율을 따져 결정한다. 인덱스를 설정하는 것 또한 이 단계에 해당한다.

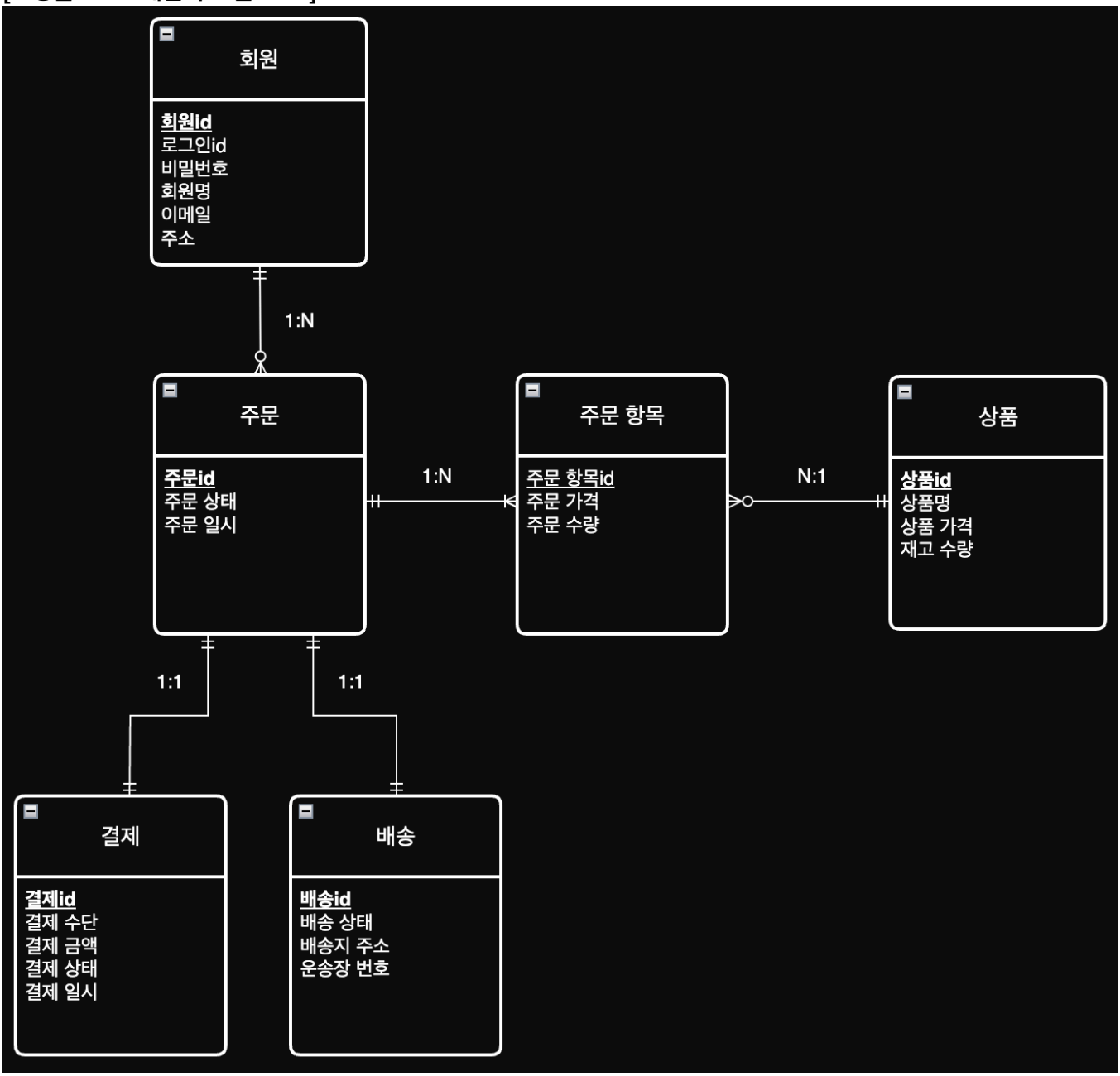
이론과 실무의 차이: 왜 경계가 모호할까?
데이터베이스 설계에서 **논리적 모델링**(청사진)과 **물리적 모델링**(실제 구현)은 이론적으로 구분되지만, 실무에서는 **개발 속도를 높이고 불필요한 수정을 줄이며, 처음부터 성능을 고려**하기 위해 두 단계를 함께 진행하는 경우가 많다. 즉, 효율성을 위해 청사진을 그리는 동시에 실제 사용할 데이터베이스의 특성을 미리 반영한다.
여기서도 컬럼과 같은 일부 내용은 논리적 모델링 단계에 함께 적용하겠다.

실전 논리적 모델링 - ERD 작성

개념적 모델 확인

먼저 우리가 완성했던 개념적 모델의 ERD를 다시 한번 확인하자. 이 청사진이 논리적 모델링의 출발점이다.

[쇼핑몰 MVP 개념적 모델 ERD]



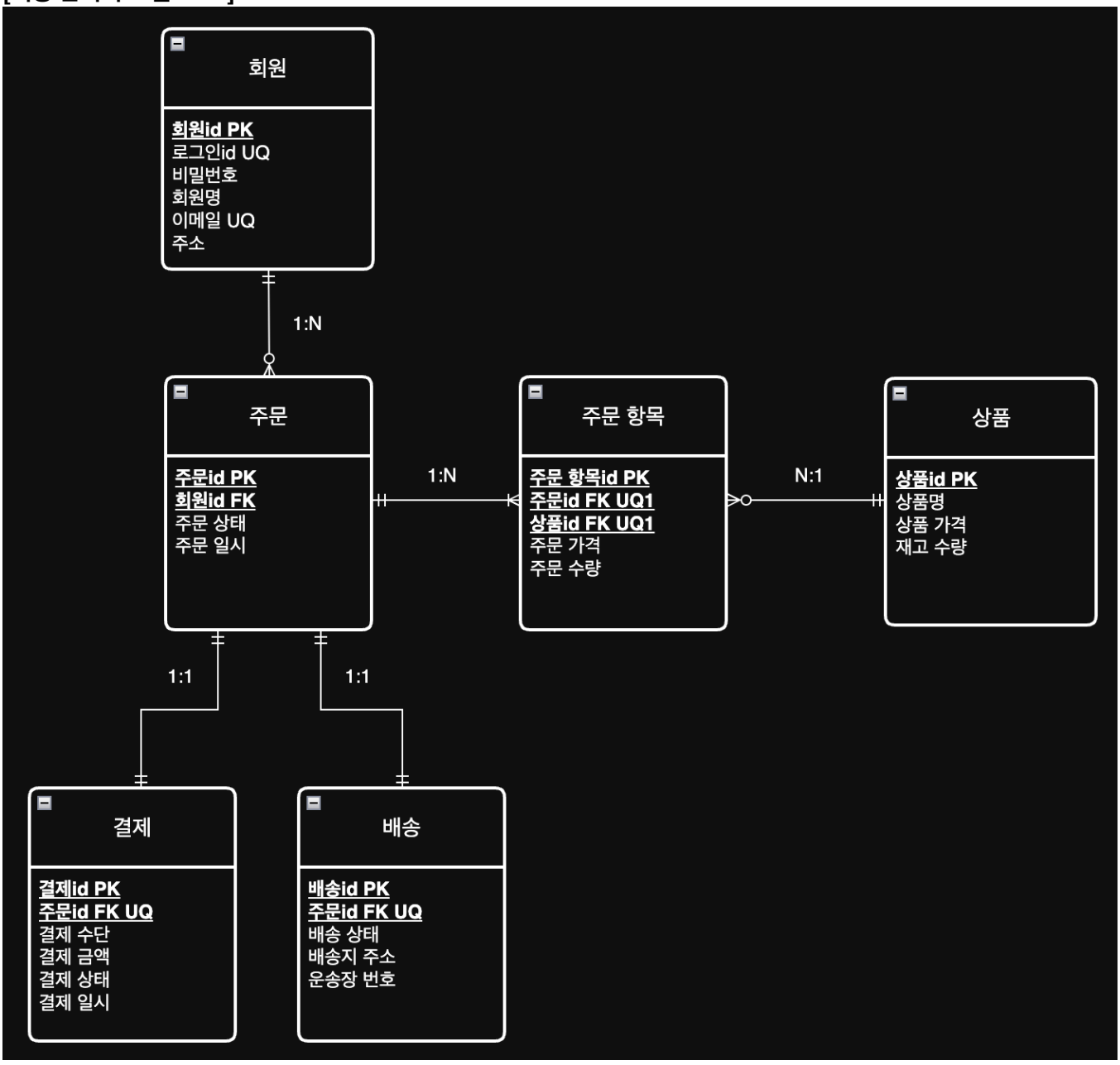
- **엔티티**: 회원, 주문, 결제, 배송, 상품
- **연관 엔티티**: 주문 항목
- **관계**: 회원 - 주문 (1:N), 주문 - 결제 (1:1), 주문 - 배송 (1:1), 주문 - 상품 (M:N)

이제 이 6개의 엔티티를 하나씩 테이블로 변환하는 작업을 시작한다.

최종 논리적 모델 ERD

최종 논리적 모델 ERD를 먼저 확인하고, 어떻게 바꾸었는지 하나씩 알아가보자.
논리적 모델로 설계한 내용을 ERD로 시각화하면 다음과 같다. 개념적 모델의 추상적인 선들이 이제 PK와 FK라는 구체적인 연결고리로 바뀌었다.

[최종 논리적 모델 ERD]



> **한글과 영문 표기 시점**
> 개념적 모델링 단계는 모든 이해관계자가 쉽게 이해할 수 있는 용어를 사용하는 것이 좋다. 따라서 한글을 사용하는 것이 좋다.
> 물리적 모델링 단계는 실제 데이터베이스 테이블로 만들어야 하기 때문에 반드시 영문으로 표기해야 한다.
> 그 중간에 있는 논리적 모델링 단계는 한글을 사용하거나 또는 한글과 영문을 병기해서 함께 표기하기도 한다.

> 논리적 모델링 단계에서도 한글을 사용하는 이유는 실무에서는 물리적 모델링과 논리적 모델링을 함께 진행하는 경우가 많은데, 이 경우 기획자를 포함한 이해관계자들이 모델을 쉽게 읽을 수 있어야 하기 때문이다.

1단계: 핵심 엔티티 테이블 설계

가장 기본이 되는 `회원(member)`과 `상품(product)` 테이블부터 설계한다. 이들은 다른 테이블의 부모가 되는 경우가 많으므로 먼저 정의하는 것이 좋다.

1. `회원` 테이블

회원 정보를 저장하는 테이블이다.

- **테이블명**: `회원`
- **기본 키(PK)**: `회원id` (대리 키)
- **컬럼 정의**
  - `회원id`: `BIGINT`, PK. 시스템이 자동으로 생성하는 고유 번호.
  - `로그인id`: `VARCHAR(50)`, 사용자가 로그인 시 사용하는 아이디. **(UNIQUE 제약조건)**
  - `비밀번호`: `VARCHAR(255)`, 해시화하여 저장될 비밀번호.
  - `회원명`: `VARCHAR(50)`, 회원명.
  - `이메일`: `VARCHAR(100)`, 이메일 주소. **(UNIQUE 제약조건)**
  - `주소`: `VARCHAR(255)`, 기본 배송지 주소.

- **설계 노트**
  - `로그인id`와 `이메일`은 비즈니스적으로 유일해야 하는 자연 키 후보들이다. 우리는 이들에 `UNIQUE` 제약조건을 걸어 데이터베이스 레벨에서 중복을 원천적으로 방지한다.
  - 기본 키는 `회원id`라는 비즈니스와 무관한 대리 키를 사용한다. 덕분에 나중에 회원이 `로그인id`나 `이메일`을 변경하더라도, 이 회원을 참조하는 수많은 주문(`orders`) 데이터는 아무런 영향을 받지 않고 안정적으로 유지된다.

2. `상품` 테이블

판매할 상품 정보를 저장하는 테이블이다.

- **테이블명**: `상품`
- **기본 키(PK)**: `상품id` (대리 키)
- **컬럼 정의**
  - `상품id`: `BIGINT`, PK. 시스템이 자동으로 생성하는 고유 번호.
  - `상품명`: `VARCHAR(100)`, 상품명.

- `상품 가격`: `INT`, 상품 가격.
- `재고 수량`: `INT`, 재고 수량.

2단계: 관계를 포함한 테이블 설계 (1:N, M:N)

이제 관계를 맺고 있는 `주문(orders)`과 `주문 항목(order_item)` 테이블을 설계한다.

3. `주문` 테이블

회원의 주문 정보를 저장한다. `회원` 테이블을 참조하는 자식 테이블이다.

- **테이블명**: `주문`
- **기본 키(PK)**: `주문id` (대리 키)
- **외래 키(FK)**: `회원id`
- **컬럼 정의**
  - `주문id`: `BIGINT`, PK. 시스템이 자동으로 생성하는 고유 번호.
  - `회원id`: `BIGINT`, FK. `member(회원id)`를 참조. 어떤 회원이 주문했는지 알려준다. NOT NULL
  - `주문 상태`: `VARCHAR(20)`, 주문 상태 (예: `ORDERED`, `SHIPPED`, `COMPLETED`, `CANCELED`).
  - `주문 일시`: `DATETIME`, 주문한 날짜 시간.

- **설계 노트**
  - `회원id` 컬럼은 `member` 테이블의 `회원id`를 참조하는 외래 키다. 이 관계는 **비식별 관계**다. 왜냐하면 `orders` 테이블은 `회원id`를 기본 키의 일부로 사용하지 않고, 자신만의 독립적인 기본 키 `주문id`를 가지고 있기 때문이다.
  - 주문은 반드시 특정 회원이 해야 하므로, `회원id`에 `NOT NULL` 제약조건을 추가하여 회원 없는 유령 주문이 생기지 않도록 강제한다. (필수 참여)

4. `주문 항목` 테이블

주문과 상품의 다대다(M:N) 관계를 해소하기 위한 **연결 테이블(Junction Table)**이다.

- **테이블명**: `주문 항목`
- **기본 키(PK)**: `주문 항목id` (대리 키)
- **외래 키(FK)**: `주문id`, `상품id`
- **컬럼 정의**
  - `주문 항목id`: `BIGINT`, PK. 시스템이 자동으로 생성하는 고유 번호.
  - `주문id`: `BIGINT`, FK. `orders(주문id)`를 참조. **(UNIQUE1 제약조건)**, NOT NULL
  - `상품id`: `BIGINT`, FK. `product(상품id)`를 참조. **(UNIQUE1 제약조건)**, NOT NULL

    - `주문 가격`: `INT`, **주문 당시의 상품 가격 (스냅샷)**.
    - `주문 수량`: `INT`, 주문한 수량.

- **설계 노트**
    - 이 테이블의 설계는 현대적 설계의 핵심을 보여준다. `(주문id, 상품id)` 복합키를 기본 키로 사용하는 식별 관계 대신, 별도의 대리 키 `주문 항목id`를 기본 키로 선택했다.
    - `주문` 및 `상품`과의 관계는 모두 **비식별 관계**가 되어 모델이 단순해지고, ORM 프레임워크와의 호환성도 극대화된다.
    - '하나의 주문에 동일 상품은 중복될 수 없다'는 비즈니스 규칙은 `UNIQUE KEY(주문id, 상품id)` 제약조건을 추가하여 보장할 것이다. 두 컬럼에 **UNIQUE1**이라고 표현해서 두 컬럼이 같은 제약조건에 묶이는 것을 표현했다.
    - `주문 가격`은 `상품` 테이블의 현재 가격을 참조하지 않고, 주문이 일어난 시점의 가격을 복사해서 저장한다. 덕분에 나중에 상품 가격이 인상되어도 과거 주문 내역의 정확성은 절대 변하지 않는다.

3단계: 1:1 관계 테이블 설계

개념적 모델에 있었던 `배송(delivery)`과 `결제(payment)`를 설계한다. 둘 다 `주문(orders)`과 1:1 관계를 맺는다.

5. `배송` 테이블

주문에 대한 배송 정보를 저장한다.

- **테이블명**: `배송`
- **기본 키(PK)**: `배송id` (대리 키)
- **외래 키(FK)**: `주문id`
- **컬럼 정의**
    - `배송id`: `BIGINT`, PK.
    - `주문id`: `BIGINT`, FK. `orders(주문id)`를 참조. **(UNIQUE 제약조건)**, NOT NULL
    - `배송 상태`: `VARCHAR(20)`, 배송 상태 (예: `READY`, `SHIPPING`, `COMPLETED`).
    - `배송지 주소`: `VARCHAR(255)`, 배송지 주소.
    - `운송장 번호`: `VARCHAR(50)`, 운송장 번호.

- **설계 노트**
    - 1:1 관계를 구현하는 가장 좋은 방법은 보조 테이블(`배송`)이 주 테이블(`주문`)을 참조하도록 외래 키를 두는 것이다.
    - `주문id` FK에 `UNIQUE` **제약조건**을 추가하는 것이 핵심이다. 이 제약조건이 `배송` 테이블에 동일한 `주문`

`id`가 두 번 이상 들어오는 것을 막아주어 완벽한 1:1 관계를 데이터베이스 레벨에서 보장한다.

6. `결제` 테이블

주문에 대한 결제 정보를 저장한다.

- **테이블명**: `결제`
- **기본 키(PK)**: `결제id` (대리 키)
- **외래 키(FK)**: `주문id`
- **컬럼 정의**
  - `결제id`: `BIGINT`, PK.
  - `주문id`: `BIGINT`, FK. `orders(주문id)`를 참조. **(UNIQUE 제약조건)**, NOT NULL
  - `결제 수단`: `VARCHAR(50)`, 결제 수단 (예: `CREDIT_CARD`).
  - `결제 금액`: `INT`, 결제 금액.
  - `결제 상태`: `VARCHAR(20)`, 결제 상태 (예: `PAID`, `FAILED`).
  - `결제 일시`: `DATETIME`, 결제 일시. 은행, 신용카드 회사의 결제 시간을 저장한다.

실무 이야기: 모델의 단순화와 트레이드 오프

지금까지 설계한 모델은 '주문', '배송', '결제'의 역할을 명확하게 분리한, 좋은 설계다. 각자의 책임이 명확하여 향후 기능 확장에 매우 유리하다.

하지만 우리 프로젝트는 이제 막 시작하는 스타트업의 MVP(최소 기능 제품) 버전이다. 초기 단계에서는 배송과 결제 시스템이 복잡하지 않을 수 있다. 이런 경우, 실무에서는 빠른 개발 속도와 단순성을 위해 일부 테이블을 합치는 전략적인 선택을 하기도 한다.

예를 들어, `배송`의 핵심 정보인 `배송지 주소`와 `배송_상태`를 `주문` 테이블에 직접 포함시켜 `배송` 테이블을 생략할 수 있다. 이렇게 하면 `JOIN` 없이 `주문` 테이블만 조회하여 주문과 배송 정보를 한 번에 볼 수 있다는 장점이 있다.

하지만 이는 **'정규화'**를 희생하여 **'단순성'**을 얻는 트레이드 오프 관계다. 서비스가 성장하여 배송 로직이 복잡해지면, 결국 `주문` 테이블에서 배송 관련 컬럼들을 다시 분리해야 하는 날이 올 수 있다. 우리는 일단 원칙에 충실한 설계를 배우고, 상황에 따라 이런 트레이드 오프를 고려할 수 있는 시야를 갖추는 것이 중요하다.

정리

이제 우리는 비즈니스 요구사항으로부터 출발하여, 유연하고 확장 가능하며, 유지보수하기 좋은 논리적 데이터 모델을 완성했다. 각 테이블은 자신의 역할에만 충실하도록 잘 분리되었고, 관계는 단순하며 명확하다.

다음 시간에는 이 논리적 모델의 품질을 한 단계 더 끌어올려 데이터 중복을 최소화하고 무결성을 극대화하는 과정인 **'정규화(Normalization)'**에 대해 알아보자.