2. 데이터베이스 소개

#0.강의/2.데이터베이스로드맵/1.입문

데이터와 정보

어떤 개념을 학습할 때 그 개념의 핵심 단어를 먼저 이해하는 것이 중요하다.
데이터베이스(Database)를 제대로 이해하려면, 먼저 그 이름의 주인공인 **데이터(Data)**라는 개념과 친해져야 한다.
데이터는 언뜻 쉬워 보이지만, **정보(Information)**라는 단어와 비교할 때 그 의미가 비로소 명확해진다.

이 둘의 차이를 이해하는 것은, 우리가 왜 데이터를 힘들게 모으고 관리하며, 데이터베이스라는 강력한 도구를 배워야 하는지에 대한 가장 근본적인 답을 준다.

데이터와 정보, 무엇이 다른가?

이제 막 시작한 우리 쇼핑몰이 제법 인기를 얻어 정신없이 바쁘다고 상상해보자. 고객 주문이 쏟아지고, 상품이 팔려나가며, 새로운 회원들이 계속 가입한다. 이 과정에서 수많은 기록, 즉 **데이터**가 쌓인다.

하지만 이 기록들을 그냥 쌓아두기만 한다면, 우리 쇼핑몰이 잘 성장하고 있는지, 어떤 상품이 가장 인기가 많은지, 누가 우리의 VIP 고객인지 알 수 있을까? 아마 어려울 것이다. 이것이 바로 데이터를 **정보**로 만드는 과정이 필요한 이유다.

1단계: 데이터(Data) - 의미 없는 기록 조각
데이터(Data)란, 아직 가공되지 않은 **개별적인 사실이나 값** 그 자체를 의미한다. 문맥(Context)이 없어서 그 자체만으로는 온전한 의미를 갖기 어려운, 말 그대로 '날것(Raw)'의 기록 조각이다.

우리 쇼핑몰의 주문 기록에서 몇 개만 떼어내 보면 다음과 같다.

- `user01`

- `티셔츠`
- `25,000`
- `2025-05-30`

이 조각들만 봐서는 `user01`이 누구인지, `티셔츠`를 산 건지 반품한 건지, `25,000`이 가격인지 재고 수량인지 알 길이 없다. 이처럼 의미가 연결되지 않는 데이터만으로는 어떤 유용한 판단도 할 수 없다.

1.5단계: 구조화된 데이터(Structured Data) - 의미를 담은 재료
현실에서 데이터는 이전처럼 완전히 흩어져 있기보다는, 수집 단계부터 **최소한의 구조를 가지고 정리**된다. 즉, 각 데이터 조각에 '꼬리표(Label)'를 붙여주는 단계다. 이는 정보라는 '요리'를 만들기 위해 재료를 손질해서 잘 정리해두는 과정과 같다.

우리 쇼핑몰의 주문 기록이 아래와 같이 표 형태로 정리되었다고 가정해보자. 이것이 바로 **구조화된 데이터**다.

주문번호	고객ID	상품명	수량	주문금액	주문일자
1001	user01	티셔츠	2	50,000	2025-05-29
1002	user02	청바지	1	70,000	2025-05-29
1003	user01	티셔츠	1	25,000	2025-05-30
1004	user03	운동화	1	120,000	2025-05-30
1005	user02	티셔츠	3	75,000	2025-05-30

1단계의 파편화된 값들과 비교하면 어떠한가? 이제 `70,000`은 '주문금액'이라는 의미를, `user01`은 '고객ID'라는 의미를 갖게 되었다. 훨씬 명확해졌다.

하지만 이 표는 여전히 '기록의 목록'일 뿐이다. 이 자체만으로는 "어떤 상품이 가장 인기가 많지?" 와 같은 질문에 바로 답해주지 않는다. 즉, **의미**는 생겼지만 의사결정에 필요한 **통찰(Insight)**은 아직 없다.

2단계: 정보(Information) - 목적을 가진 결과물
정보(Information)는 이렇게 구조화된 데이터를 **특정한 질문이나 목적을 가지고 분석하고 가공해서 얻어낸 유의미한 결과물**이다. 데이터라는 '재료'를 가지고 질문에 대한 답을 찾아낸 '완성된 요리'인 셈이다.

쇼핑몰 대표님이 우리에게 이렇게 질문했다고 가정해보자.
"2025년 5월 30일에 가장 많이 팔린 상품은 무엇이고, 총 몇 개나 팔렸는가?"

이 질문에 답하기 위해, 우리는 1.5단계의 '구조화된 데이터'를 다음과 같이 가공한다.

1. **필터링(Filtering):** '주문일자'가 '2025-05-30'인 데이터만 골라낸다.
    - 주문 1003, 1004, 1005가 해당된다.
2. **그룹화 및 집계(Grouping & Aggregation):** 골라낸 데이터를 '상품명'으로 묶고, '수량'을 합산한다.
    - 티셔츠: 1개 (1003) + 3개 (1005) = **총 4개**
    - 운동화: **총 1개** (1004)

이제, 우리는 대표님의 질문에 대한 명확한 **정보**를 드릴 수 있다.

- "2025년 5월 30일에는 '티셔츠'가 총 4개로 가장 많이 팔렸습니다."

이 외에도 우리는 같은 데이터를 활용해서 아래와 같은 다양한 정보를 뽑아낼 수 있다.

- "고객 'user01'은 5월 29일과 30일 모두 구매한 **재구매 고객**이다."
- "5월 30일의 **총 매출액**은 220,000원이다."
- "가장 높은 금액의 단일 주문은 **120,000원짜리 운동화**였다."

그래서, 이게 왜 중요한가?

우리 쇼핑몰이 성공하려면 위와 같은 정보를 바탕으로 똑똑한 결정을 내려야 한다.

- 어떤 상품이 잘 팔리는지 (정보) 알아야 재고 관리를 효율적으로 할 수 있다 (의사결정).
- 누가 우리의 VIP 고객인지 (정보) 알아야 맞춤형 마케팅을 제공할 수 있다 (의사결정).
- 날짜별 매출 추이가 어떤지 (정보) 알아야 새로운 프로모션을 기획할 수 있다 (의사결정).

이 모든 것은 흩어져 있는 **데이터**를 의미 있는 **정보**로 만들 때 비로소 가능해진다.

그리고 우리가 앞으로 배울 **데이터베이스**는 바로 이 데이터를 체계적으로 저장하고(1.5단계), 우리가 원하는 조건으로 쉽고 빠르게 찾아내고 가공해서(2단계) 정보로 만들 수 있도록 도와주는 가장 강력한 핵심 도구다.

> **베이스(base)**: "기반, 토대, 기지"라는 의미로, 여기서는 데이터가 체계적으로 저장되고 관리되는 "저장소" 또는 "기반 시설"을 나타낸다. 마치 군사 기지가 군사 활동의 거점이 되듯이, 데이터베이스는 데이터 관리의 중심 거점 역할을 한다.
> 정리하면 데이터베이스는 **"데이터를 체계적으로 모아 놓은 핵심 저장소"**라는 뜻이다.

데이터베이스 관리 시스템이 필요한 이유

우리가 쇼핑몰을 운영하면서 생기는 그 수많은 데이터들, 예를 들어 고객 정보, 상품 정보, 주문 내역 같은 것들을 어딘가에는 저장해야 정보로 만들 수 있다. 이런 데이터들을 그냥 머릿속에 기억해 둘 수는 없다.

왜 우리는 데이터 저장 방식에 대해 고민해야 할까?

쇼핑몰이 점점 커지면서 데이터는 기하급수적으로 늘어난다. 처음에는 몇 개 안 되는 주문이라 손으로 메모장이나 장부에 적을 수도 있다. 하지만 하루에 수백, 수천 건의 주문이 들어오고, 수만 명의 고객 정보를 관리해야 한다면? 그때도 주먹구구식으로 데이터를 관리할 수 있을까? 당연히 어렵다. 데이터가 엉망으로 관리되면, 필요한 정보를 제때 얻을 수 없고, 결국 우리 쇼핑몰 운영에 큰 차질이 생긴다. 그래서 데이터를 '어떻게 잘 보관하고 관리할 것인가'는 아주 중요한 문제가 된다.

데이터를 어디에 보관할 것인가?

가장 원시적인 방법부터 생각해 보자.

- **기억**: 사람의 기억력은 한계가 있고 부정확하다. 우리 쇼핑몰의 모든 거래를 내가 다 기억할 수는 없다.
- **종이 문서**: 초창기 작은 가게라면 가능할 수 있다. 주문서, 고객 명부 등을 수기로 작성한다. 하지만 데이터가 많아지면 책상이 서류로 뒤덮일 거고, 특정 내용을 찾으려면 하루 종일 종이만 뒤적거려야 할 것이다. 분실 위험도 크고, 수정도 어렵고, 여러 사람이 동시에 보기도 힘들다.

컴퓨터 파일 시스템을 이용한 데이터 관리

그럼 컴퓨터를 활용하는 방법을 생각해보자. 가장 먼저 떠오르는 건 아마 **파일(File) 시스템**일 것이다.

예시: 텍스트 파일(txt)로 데이터를 관리하는 경우

가장 단순한 텍스트 파일(메모장)로 데이터를 관리해보자.

- 고객 정보는 `customers.txt` 같은 파일에 저장하고,
- 상품 목록은 `products.txt` 파일에 저장하고,
- 주문 내역은 `orders_202505.txt` 같은 파일에 날짜별로 저장하는 방식이다.

고객 정보 파일명: `customers.txt`

```
C001,네이트,010-1234-0001,서울시 강남구 테헤란로 123
C002,션,010-1234-0002,서울시 서초구 강남대로 123
C003,김수로,010-1234-0003,서울시 서초구 강남대로 777
```

상품 목록 파일명: `products.txt`

```
P01,프리미엄 키보드,150000,50
P02,고성능 마우스,80000,120
P03,QHD 27인치 모니터,350000,30
```

주문 내역 파일명: `orders_202505.txt`

```
001/2025-05-30/C001/P01/1
002/2025-05-31/C003/P02/2
```

위와 같이 텍스트 파일은 간단하게 정보를 나열할 수 있지만, 데이터의 구조가 명확하지 않고, 특정 데이터를 찾거나 수정하기가 어렵다. 콤마(,)나 슬래시(/) 같은 특정 문자로 데이터를 구분해야만 한다.

예시: 엑셀(스프레드시트)로 데이터를 관리하는 경우

텍스트 파일에 저장하는 것보다는 좀 더 구조화된 엑셀 형식의 파일로 저장하고 관리하는 것이 더 나은 방법이다.

- 고객 정보는 `customers.xlsx` 파일에 저장하고,
- 상품 목록은 `products.xlsx` 파일에 저장하고,
- 주문 내역은 `orders_202505.xlsx` 같은 파일에 날짜별로 저장하는 방식이다.

파일명: `customers.xlsx`

고객ID	이름	전화번호	주소
C001	네이트	010-1234-0001	서울시 강남구 테헤란로 123
C002	션	010-1234-0002	서울시 서초구 강남대로 123
C003	김수로	010-1234-0003	서울시 서초구 강남대로 777

파일명: `products.xlsx`

상품ID	상품명	가격	재고
P01	프리미엄 키보드	150,000	50

P02	고성능 마우스	80,000	120
P03	QHD 27인치 모니터	350,000	30

파일명: `orders_202505.xlsx`

주문번호	주문일	고객ID	상품ID	수량
001	2025-05-30	C001	P01	1
002	2025-05-31	C003	P02	2

엑셀은 행(Row)과 열(Column)로 데이터가 명확하게 구조화되어 있어 텍스트 파일보다 훨씬 보기 편하고 관리하기 좋다. 그리고 정렬, 필터링, 간단한 계산 등 다양한 기능도 활용할 수 있다.

실제로 작은 규모의 작업이나 개인 프로젝트에서는 이렇게 파일 기반으로 데이터를 관리하기도 한다. 초기에는 이게 편해 보일 수도 있다. 하지만 우리 쇼핑몰처럼 규모가 점점 커지고 데이터가 복잡해지기 시작하면, 엑셀 형식을 포함한 파일 시스템은 여러 가지 심각한 문제점들을 드러내기 시작한다.

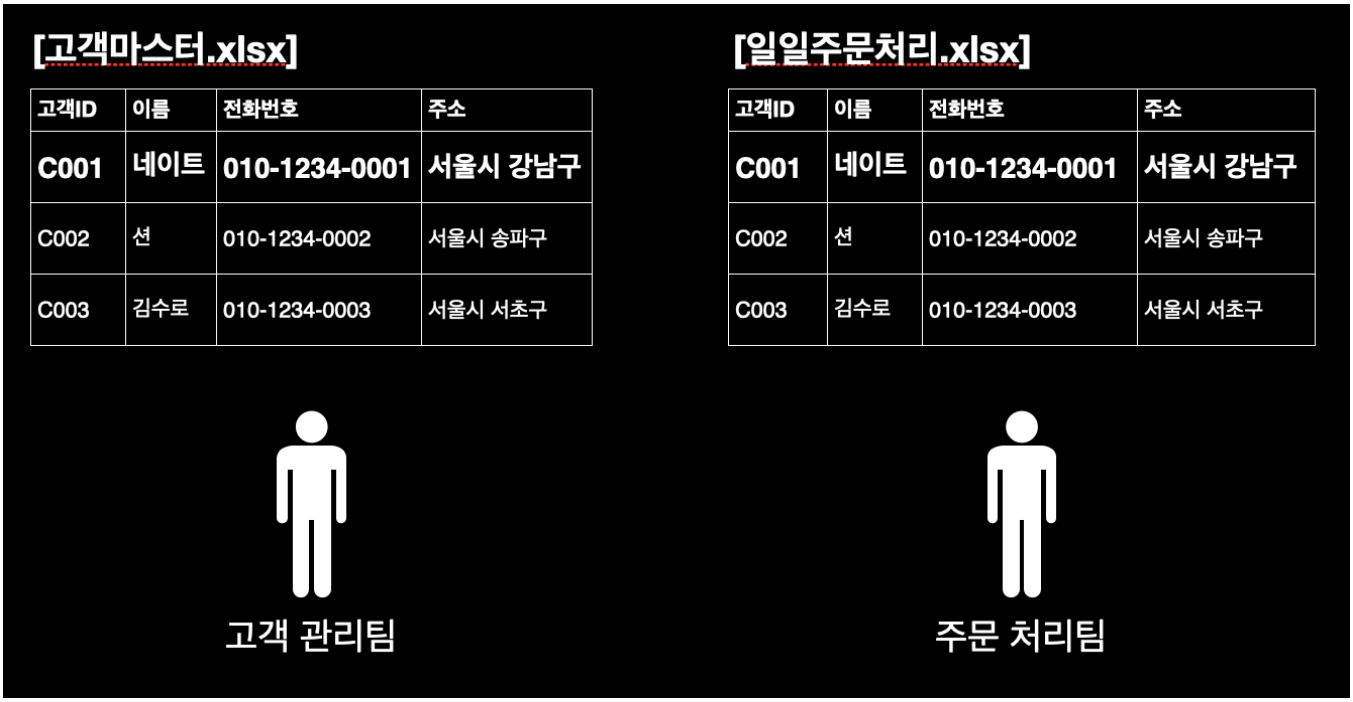
파일에 직접 저장하고 관리하는 것의 문제점

파일에 직접 데이터를 저장하고 관리할 때 어떤 문제들이 발생하는지, 우리 쇼핑몰 사례를 통해 하나씩 자세히 살펴보자.

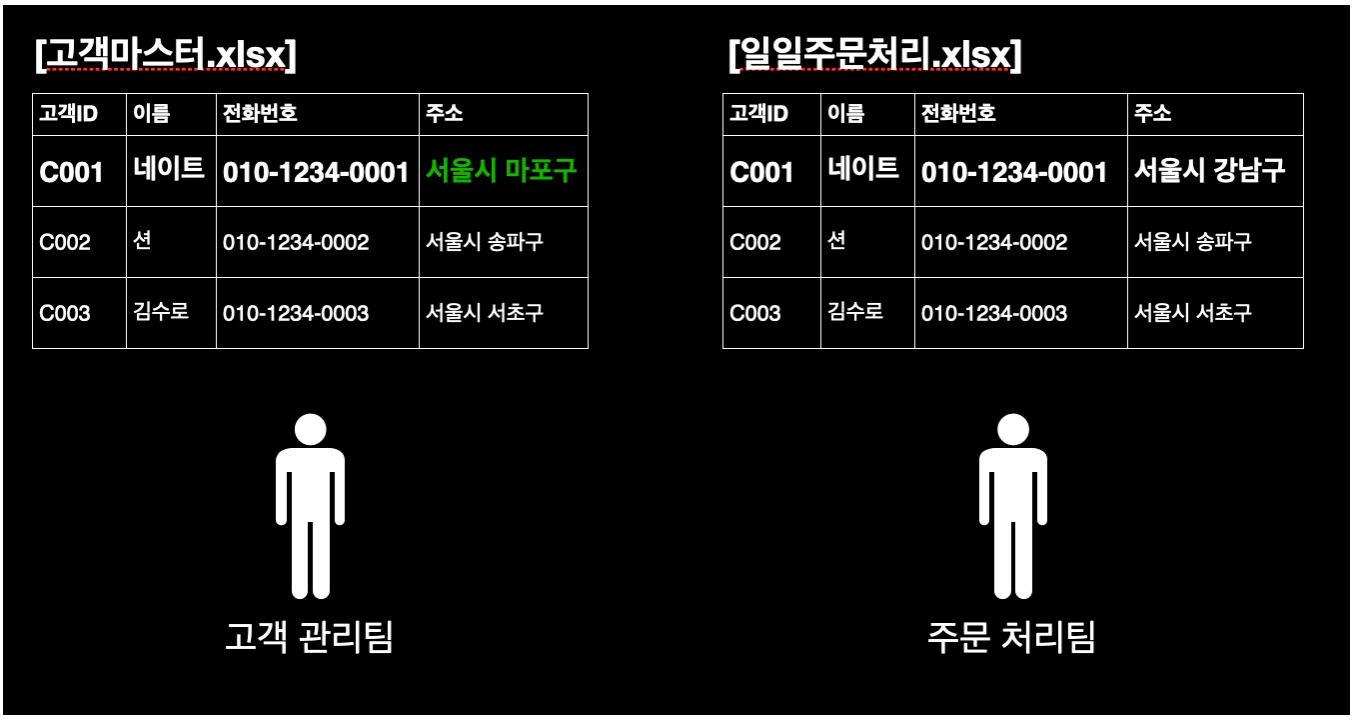
1. 데이터 중복과 불일치

문제 상황: 우리 쇼핑몰의 규모가 커지면서, **고객 관리팀**과 **주문 처리팀**이 각자 업무에 필요한 엑셀 파일을 따로 관리하기 시작했다고 가정해 보자.

- **고객 관리팀**은 고객의 전체 정보를 담은 `고객마스터.xlsx` 파일을 관리한다. 여기에는 고객의 주소, 연락처, 등급 등의 정보가 있다.
- **주문 처리팀**은 매일 들어오는 주문을 처리하기 위해 `일일주문처리.xlsx` 파일을 사용한다. 신속한 업무를 위해 자주 주문하는 고객의 정보(이름, 주소, 연락처)를 이 파일에 복사해두고 사용한다.



어느 날, 단골 고객인 '네이트'님이 이사를 해서 고객센터에 주소 변경을 요청했다. 고객 관리팀 직원은 `고객마스터.xlsx` 파일에 있는 네이트 고객의 주소를 '서울시 강남구'에서 '서울시 마포구'로 정확하게 수정했다.



며칠 후, 네이트 고객이 새로운 주문을 넣었다. 그런데 주문 처리팀 직원은 바쁜 나머지 `고객마스터.xlsx` 파일을 확인하는 대신, 자신들의 `일일주문처리.xlsx` 파일에 저장되어 있던 기존의 '서울시 강남구' 주소를 그대로 사용하여 배송을 처리했다.

결과

- **데이터 중복(Redundancy):** '네이트' 고객의 주소 정보가 `고객마스터.xlsx`와 `일일주문처리.xlsx` 두 개의 파

일에 중복으로 저장되어 있다.

- **데이터 불일치(Inconsistency):** 이제 '네이트' 고객의 주소는 `고객마스터.xlsx`에는 **'서울시 마포구'**로, `일일주문처리.xlsx`에는 **'서울시 강남구'**로 기록되어, 동일한 고객에 대한 정보가 서로 다른 값을 갖게 되었다.
  결국 이 주문은 잘못된 주소로 배송되었고, 이는 배송 지연, 고객의 불만 제기, 상품 회수 및 재배송에 드는 추가 비용 발생 등 심각한 문제로 이어진다. 어떤 주소가 진짜 맞는 주소인지 확인하기 위해 여러 파일을 뒤져봐야 하는 혼란도 발생한다.
- **실무에서는:** 이처럼 데이터가 여러 파일에 나뉘어 중복 저장되면 저장 공간 낭비는 물론, 하나의 정보가 수정될 때마다 관련된 모든 파일을 찾아 빠짐없이 수정해야 하는 번거로움이 생긴다. 담당자가 실수로 하나라도 놓치면 데이터는 어긋나 버리고, 이는 곧 회사의 신뢰도 하락과 금전적 손실로 직결된다.

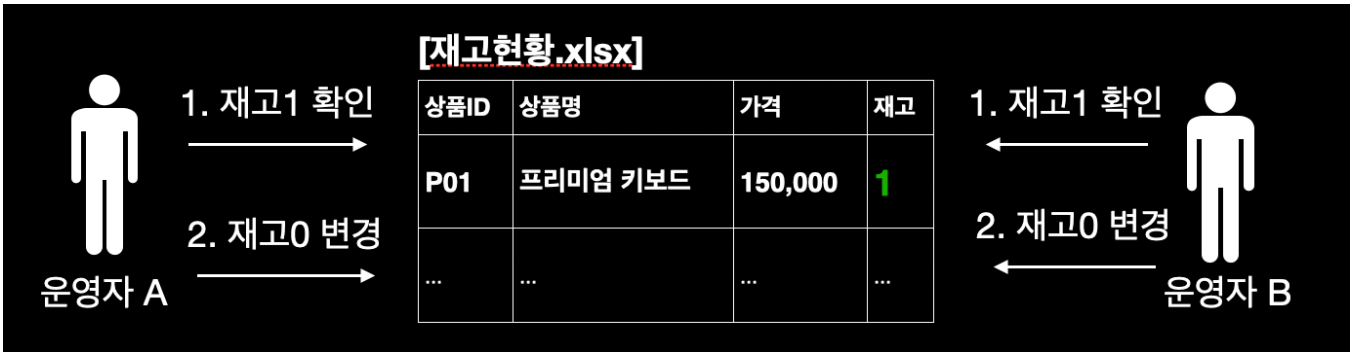
2. 데이터 접근의 어려움

- **문제 상황:** 마케팅팀에서 "지난달에 '서울시 강남구'에 거주하는 고객들 중 '운동화'를 구매한 고객 명단을 뽑아주세요. 그리고 그 고객들의 총 구매액도 알려주세요." 라고 요청했다고 해보자. 만약 우리가 고객 정보 파일, 상품 정보 파일, 주문 내역 파일을 각각 따로 가지고 있다면 이 작업을 어떻게 해야 할까?
- **결과:** 먼저 주문 내역 파일들을 전부 뒤져서 지난달 '운동화' 구매 기록을 찾아야 한다. 그리고 각 주문의 고객 ID를 가지고 고객 정보 파일에서 '서울시 강남구' 거주 고객인지 확인해야 한다. 수백, 수천 개의 파일이 있다면? 이건 거의 수작업으로는 불가능에 가깝다. 설령 프로그래밍을 할 줄 아는 개발자가 이 작업을 위한 코드를 매번 작성한다고 해도, 새로운 유형의 요청이 올 때마다 새로운 코드를 짜야 하니 매우 비효율적이다.
- **실무에서는:** 데이터가 단순히 파일 덩어리로 존재하면, 원하는 조건으로 데이터를 검색하거나 여러 파일에 흩어져 있는 데이터를 조합해서 의미 있는 정보를 얻어내는 것이 어렵고, 시간도 많이 걸린다.

3. 데이터 무결성 제약조건 적용의 어려움

- **문제 상황:** 데이터에는 일정한 규칙과 형식이 있어야 한다. 예를 들어, 우리 쇼핑몰에서 상품 가격은 항상 0보다 커야 하고, 주문 수량도 최소 1개 이상이어야 한다. 회원의 아이디는 중복되어서는 안 된다. 이런 것들을 **데이터 무결성 제약조건**이라고 한다.
- **결과:** 일반적인 파일 시스템에서는 이런 제약조건을 강제하기가 어렵다. 직원이 실수로 상품 가격에 음수(-)를 입력하거나, 주문 수량에 0을 입력해도 파일 자체는 그냥 받아들일 수 있다. 특히 아이디가 같은 사용자가 2명이라면 심각한 문제가 발생할 것이다. 이렇게 잘못된 데이터가 시스템에 쌓이면 나중에 큰 문제를 일으키게 된다. 주문 관리 프로그램을 만들어서 사용한다고 가정해도 모든 유효성 검사를 데이터를 입력하는 애플리케이션(예: 주문 처리 프로그램)에서 일일이 다 책임져야 하는데, 이건 매우 번거롭고 빠뜨리기 쉽다.
- **실무에서는:** 데이터의 정확성과 일관성을 유지하는 것이 매우 중요하다. 파일 시스템은 이를 위한 체계적인 지원이 부족하다.

4. 동시성 제어 (Concurrency Control) 문제



- **문제 상황:** 우리 쇼핑몰이 잘 돼서 여러 명의 운영자가 동시에 주문을 처리한다고 생각해보자. '프리미엄 키보드' 재고가 딱 1개 남은 상황이다.
  1. 운영자 A가 고객 전화를 받고 재고 파일(예: `재고현황.xlsx`)을 열어보니 1개가 있어서 "구매 가능하다"고 안내했다. 거의 동시에 운영자 B도 같은 파일을 열어 재고 1개를 확인하고 "구매 가능하다"고 안내했다.
  2. 운영자 A가 주문을 접수하고 재고 파일의 재고를 1 ➔ 0으로 변경한다. 동시에 운영자 B도 주문을 접수하고 재고 파일의 재고를 1 ➔ 0으로 변경한다.
- **결과:** 고객 2명은 이미 주문을 완료했는데, 실제로는 재고가 1개 밖에 없는 황당한 상황이 발생한다. 이런 문제를 **동시성 제어 문제**라고 한다. 여러 사용자가 동시에 같은 데이터에 접근해서 수정하려고 할 때 데이터의 일관성이 깨지는 것이다.
- **실무에서는:** 파일 시스템은 기본적으로 여러 사용자가 동시에 안전하게 데이터를 수정하는 것을 고려하여 설계되지 않았다. 누가 먼저 파일을 열었는지, 수정 중인 내용은 어떻게 처리할지 등에 대한 복잡한 제어 로직이 없다.

5. 보안 (Security) 문제

- **문제 상황:** 우리 고객 정보 파일에는 고객의 이름, 주소, 전화번호뿐만 아니라 민감할 수 있는 주민등록번호나 계좌 번호가 포함되어 있다고 가정해보자. 이 파일을 누가 열어볼 수 있어야 할까? 모든 직원이 다 봐도 될까? 특정 직원에게는 고객 이름과 연락처만 보여주고, 재무팀 직원에게만 계좌 번호를 보여주는 식으로 권한을 다르게 설정할 수 있을까?
- **결과:** 일반적인 파일 시스템의 접근 권한 설정(읽기, 쓰기, 실행 권한 등)만으로는 이렇게 세밀한 데이터 접근 제어가 어렵다. 파일 전체에 대한 접근 권한만 제어할 수 있을 뿐, 파일 내용의 특정 부분에 대한 접근을 사용자별로 다르게 설정하기는 거의 불가능하다. 이는 데이터 유출의 위험을 높인다.
- **실무에서는:** 중요한 비즈니스 데이터, 특히 개인 정보는 철저한 보안 관리가 필수적이다. 파일 시스템만으로는 이런 요구사항을 충족시키기 어렵다.

6. 데이터 복구(Recovery) 및 백업(Backup)의 어려움

- **문제 상황:** 만약 우리 쇼핑몰의 모든 주문 내역과 고객 정보가 저장된 하드디스크가 갑자기 고장 나면 어떻게 될까? 또는 직원이 실수로 중요한 데이터 파일을 삭제해버렸다면? 정기적으로 파일을 백업해두지 않았다면 모든 데이터를 날리는 끔찍한 상황이 발생할 수 있다.
- **결과:** 설령 백업을 해두었다고 해도, 시스템 장애 직전의 가장 최신 상태로, 그리고 데이터들 간의 일관성을 유지하면서 복구하는 것은 매우 복잡하고 어려운 작업이다. 예를 들어, 주문 처리 중에 시스템이 멈췄다면, 주문 정보는 저장되었는데 결제 정보는 저장되지 않은 불일치 상태로 복구될 수도 있다.

- **실무에서는:** 데이터는 비즈니스의 핵심 자산이므로, 어떤 장애 상황에서도 안전하게 보호되고 신속하게 복구될 수 있는 체계가 필요하다. 파일 시스템은 이를 위한 정교한 메커니즘을 제공하지 않는다.

데이터베이스 관리 시스템의 등장

지금까지 파일 시스템으로 데이터를 관리할 때 발생할 수 있는 다양한 문제점들을 살펴보았다. 데이터 중복 및 불일치, 데이터 접근의 어려움, 무결성 유지의 어려움, 동시성 제어 문제, 보안 문제, 그리고 회복과 백업의 어려움까지. 우리 쇼핑몰이 성장함에 따라 이런 문제들은 점점 더 심각해지고, 결국에는 쇼핑몰 운영 자체가 불가능해질 수도 있다.

바로 이러한 문제점들을 해결하기 위해 등장한 것이 데이터베이스 관리 시스템이다.

데이터베이스 관리 시스템은 단순히 파일에 데이터를 모아둔 것을 넘어, 데이터를 보다 체계적으로 구성하고, 앞서 언급된 문제점들을 해결하기 위한 다양한 기능들을 제공한다.

- 데이터 중복을 최소화하고 일관성을 유지하며,
- 다양하고 복잡한 조건으로도 쉽게 데이터를 검색하고 활용할 수 있게 하며,
- 데이터의 무결성 규칙을 강제하여 데이터의 품질을 높이며,
- 여러 사용자가 동시에 접근해도 데이터가 안전하게 처리되도록 제어하며,
- 정교한 보안 정책을 통해 데이터 접근을 통제하며,
- 장애 발생 시 데이터를 안전하게 복구하고 백업할 수 있는 기능을 제공한다.

즉, 데이터베이스 관리 시스템은 우리 쇼핑몰의 소중한 데이터를 안전하고 효율적으로 관리하여, 우리가 필요로 하는 정보를 언제든지 정확하게 얻을 수 있도록 도와주는 핵심 기반 시설이라고 할 수 있다.

다음 시간에는 데이터베이스 관리 시스템(DBMS)에 대해 자세히 알아보자.

데이터베이스 관리 시스템 (DBMS) 소개

지난 시간에는 파일 시스템으로 데이터를 관리할 때 발생하는 다양한 문제점들에 대해 알아보았다. 데이터가 중복되고, 일관성이 깨지고, 찾기도 어렵고, 보안도 허술하고, 여러 사람이 동시에 쓰기도 힘들고, 고장 나면 복구도 어렵고... 우리 쇼핑몰이 이런 식으로 운영된다면 정말 큰일일 것이다.

그럼 이런 문제들을 어떻게 해결할 수 있을까? **데이터베이스 관리 시스템(DBMS, Database Management System)**이 그 해결사 역할을 한다. 오늘은 이 DBMS가 도대체 무엇인지, 그리고 우리 쇼핑몰을 위해 어떤 중요한 일

들을 하는지 알아보자.

데이터베이스 용어 정리

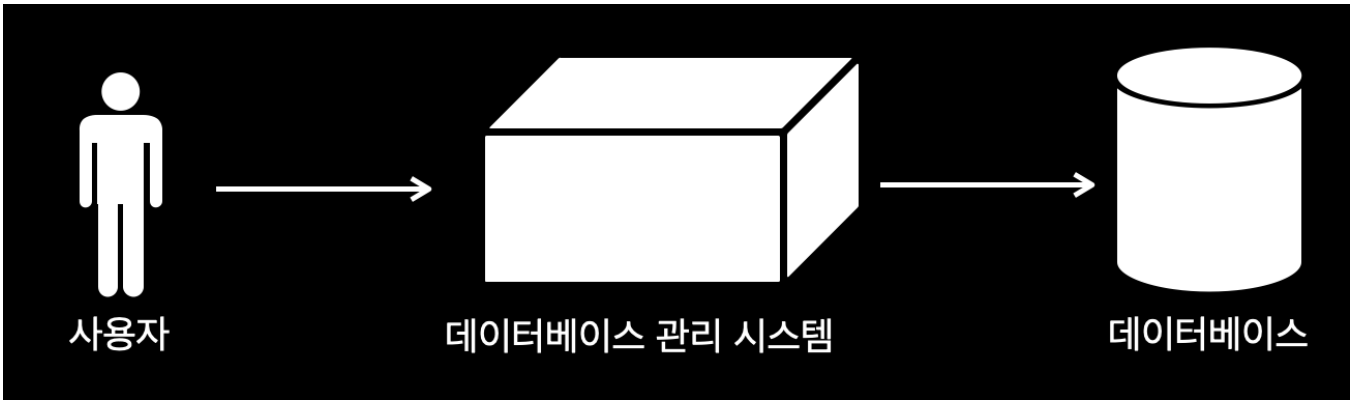
데이터베이스라는 용어는 넓은 의미로는 다음과 같이 사용된다.

데이터베이스는 특정 목적을 위해 구조화된 데이터의 모음이다. 예를 들어, 회사의 고객 정보, 제품 목록, 판매 기록 등을 엑셀 시트나 특정 형식의 파일에 정리해 둔 것 자체가 데이터베이스에 해당한다.

- **예시:**
  - 회원 정보가 담긴 표
  - 도서관의 도서 목록
  - 온라인 쇼핑몰의 상품 정보

데이터베이스 관리 시스템(DBMS)이란 무엇인가?

데이터베이스 관리 시스템(DBMS)이란, 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 **데이터베이스를 생성, 정의, 공유, 사용, 관리하는 기능을 수행하는 소프트웨어 시스템**을 말한다.



쉽게 이야기해서 데이터베이스는 데이터를 체계적으로 모아놓은 창고이고, 데이터베이스 관리 시스템(DBMS)은 **이 창고를 효율적으로 관리하는 소프트웨어**라고 할 수 있다.

파일 시스템과 DBMS 비교

DBMS를 사용하면 개발자가 직접 파일을 다루는 대신에 DBMS 소프트웨어가 대신 파일을 사용해서 데이터를 저장하고 관리한다.

1. **DBMS의 역할**: DBMS는 데이터를 효율적으로 저장, 검색, 수정, 삭제할 수 있도록 하는 소프트웨어이다. (프로그램으로 생각하면 된다. 실제로 설치해야 한다.)
2. **데이터 저장**: DBMS는 결국 데이터를 운영체제의 파일 시스템 위에 있는 **특정한 형식의 파일들**에 저장한다. 이 파일들은 일반적인 텍스트 파일이나 CSV 파일과는 달리 DBMS 자체의 특별한 구조와 방식으로 구성되어 있어서 매우 효율적으로 데이터를 관리할 수 있다.
3. **추상화**: 개발자나 사용자는 DBMS가 내부적으로 어떤 파일에 어떻게 데이터를 저장하는지 알 필요가 없다. 대신

SQL과 같은 표준화된 언어를 사용하여 DBMS에 "이런 데이터를 저장해줘" 또는 "이런 데이터를 찾아줘" 라고 요청만 하면 된다. 그러면 DBMS가 알아서 내부 파일을 조작하여 요청을 처리한다.

4. **직접 접근 금지**: DBMS가 관리하는 내부 데이터 파일들은 사용자가 직접 열어보거나 수정하려고 해서는 안 된다. 그렇게 할 경우 데이터가 손상되거나 DBMS 시스템 전체에 문제가 생길 수 있다. 모든 데이터 접근은 반드시 DBMS를 통해서 이루어져야 한다.

간단히 비유하자면:

- **파일 직접 처리**: 직접 돈을 직접 금고(파일)에 넣고 빼고 관리하는 것과 같다. 어떤 돈을 어디에 뒀는지, 안전한지 등을 직접 신경 써야 한다.
- **DBMS 사용**: 은행(DBMS)에 돈을 맡기는 것과 같다. 여러분은 은행 창구(SQL 등 인터페이스)를 통해 입출금 요청만 하면, 은행 내부에서는 어떤 금고(내부 파일)에 어떻게 돈을 보관하고 관리하는지는 알아서 처리해 준다. 여러분은 은행 내부의 복잡한 시스템을 알 필요가 없다.

결론적으로, DBMS는 개발자가 파일 시스템의 복잡한 부분을 직접 다루지 않도록 하고, 대신 **데이터를 보다 체계적이고 안전하며 효율적으로 관리할 수 있는 추상화된 방법을 제공**하는 것이다. 그리고 그 과정에서 DBMS는 내부적으로 파일을 사용해서 실제 데이터를 디스크에 저장한다.

DBMS는 도서관 사서

쉽게 비유를 들어보자. DBMS는 **거대한 도서관의 아주 유능한 사서**와 같다고 생각할 수 있다.
도서관에는 수많은 책(데이터)들이 있다. 사서(DBMS)는 이 책들을 그냥 아무렇게나 쌓아두지 않는다.

- 책의 주제별로 분류하고(데이터 정의),
- 새로운 책이 들어오면 등록하고(데이터 생성),
- 어떤 책이 어디 있는지를 목록(카탈로그)으로 만들고 관리하며(데이터 관리),
- 사람들이 책을 쉽게 찾고 빌려볼 수 있도록 도와주고(데이터 조작 및 접근),
- 아무나 귀중한 고서를 만지지 못하게 하고(데이터 제어, 보안),
- 책이 훼손되지 않도록 관리하고(데이터 무결성),
- 도서관 문을 닫은 후에도 책이 안전하게 보관되도록 하며(백업 및 복구),
- 여러 사람이 동시에 도서관을 이용해도 혼란이 없도록 질서를 유지한다(동시성 제어).

이 모든 복잡하고 중요한 일을 사서(DBMS)가 다 알아서 해주는 것이다. 우리는 그냥 사서에게 "혹시 '성공하는 쇼핑몰 운영 비법'에 대한 책 있나요?" 라고 물어보기만 하면, 사서가 알아서 찾아다 주는 것이다. 우리가 직접 수만 권의 책 사이를 헤맬 필요가 없는 것처럼, 애플리케이션 개발자나 사용자는 데이터가 디스크에 물리적으로 어떻게 저장되는지 몰라도 DBMS를 통해 논리적으로 데이터에 접근하고 활용할 수 있다.

DBMS의 역할과 기능

이 유능한 사서, 즉 DBMS가 구체적으로 어떤 역할과 기능을 수행하는지, 그리고 이것이 어떻게 파일 시스템의 문제점들을 해결하는지 하나씩 살펴보자.

1. **데이터 정의 기능 (Data Definition Language - DDL)**
    - **역할:** DBMS는 데이터베이스의 구조를 정의할 수 있는 기능을 제공한다. 우리 쇼핑몰에 어떤 데이터가 필요한지, 각 데이터는 어떤 형태(숫자, 문자열, 날짜 등)로 저장될지, 데이터들 간의 관계는 어떻게 되는지, 어떤 제약조건(예: 상품 가격은 항상 양수여야 한다, 고객 ID는 중복될 수 없다)을 가져야 하는지 등을 명시할 수 있다.
    - **예시:** 우리 쇼핑몰의 경우 `고객(Customers)` 테이블, `상품(Products)` 테이블, `주문(Orders)` 테이블 등을 만들고, 각 테이블에 어떤 항목(예: 고객 테이블에는 고객ID, 이름, 이메일, 주소 등)이 들어갈지, 각 항목의 데이터 타입(예: 고객ID는 정수형, 이름은 문자열)은 무엇인지 정의할 수 있다. 또한, '고객ID는 반드시 존재해야 한다(`NOT NULL`)', '이메일은 유일해야 한다(`UNIQUE`)' 같은 제약조건도 설정할 수 있다. 참고로 테이블은 엑셀의 표로 이해하면 된다.
    - **파일 시스템 문제 해결:** 파일 시스템에서는 이런 구조나 제약조건을 체계적으로 관리하기 어려웠던 문제를 해결해준다.

2. **데이터 조작 기능 (Data Manipulation Language - DML)**
    - **역할:** 일단 데이터베이스의 구조가 정의되면, 사용자는 이 구조에 맞춰 데이터를 등록(Insert), 조회(Select), 수정(Update), 삭제(Delete)할 수 있어야 한다. DBMS는 이러한 데이터 조작을 위한 효율적이고 편리한 방법을 제공한다. 특히 관계형 DBMS에서는 SQL(Structured Query Language)이라는 표준화된 언어를 주로 사용한다.
    - **예시:**
        - 새로운 고객이 가입하면 `고객` 테이블에 정보를 등록한다.
        - "지난달 가장 많이 팔린 상품 Top 5"를 알고 싶을 때, `주문` 테이블과 `상품` 테이블에서 조건에 맞는 데이터를 조회한다.
        - 고객이 이사를 가서 주소가 바뀌면 `고객` 테이블의 주소 정보를 수정한다.
        - 단종된 상품 정보를 `상품` 테이블에서 삭제한다.
    - **파일 시스템 문제 해결:** 파일 시스템에서는 특정 조건에 맞는 데이터를 찾거나 여러 파일의 데이터를 조합하는 것이 매우 복잡했던 문제를 SQL과 같은 강력한 언어를 통해 매우 쉽게 해결해준다.

3. **보안, 동시성 제어, 트랜잭션 관리 기능**
    - **역할:** DBMS는 데이터의 보안을 유지하고, 여러 사용자가 동시에 데이터에 접근할 때 발생할 수 있는 문제를 제어하는 기능을 수행한다.
        - **보안(Security):** 허가된 사용자만이 데이터에 접근할 수 있도록 하고, 사용자별로 접근 가능한 데이

터의 범위나 수행할 수 있는 작업(읽기, 쓰기, 수정 등)을 제한할 수 있다.

- **동시성 제어(Concurrency Control):** 여러 사용자가 동시에 같은 데이터를 수정하려고 할 때, 데이터의 일관성이 깨지지 않도록 순서를 제어하거나 잠금(Locking) 메커니즘을 사용한다. (예: 쇼핑몰에서 동시에 여러 고객이 마지막 남은 한 개의 상품을 주문하려고 할 때, 한 고객에게만 판매가 완료되도록 처리)
- **트랜잭션 관리(Transaction Management):** 아주 중요한 개념인데, 여러 개의 작업을 하나의 논리적인 단위(트랜잭션)로 묶어서 처리한다. 이 트랜잭션은 전부 성공적으로 완료되거나(Commit), 하나라도 실패하면 전부 이전 상태로 되돌아가야(Rollback) 데이터의 일관성이 보장된다 (이를 **원자성(Atomicity)** 이라고 한다). 예를 들어, 계좌 이체라는 것은 (1) 나의 돈을 출금하고, (2) 대상 계좌에 돈을 입금하는 과정이다. 만약 나의 돈을 출금했는데, 대상 계좌에 돈을 입금하는 과정에서 문제가 발생하게 되면, 나의 돈만 사라지는 심각한 문제가 발생한다. 따라서 이 두 작업은 모두 성공하거나 모두 실패해야 한다. 만약 대상 계좌에 돈을 입금하는 과정에서 문제가 발생하면 나의 돈을 출금했던 것도 취소되어야 한다. DBMS는 이러한 트랜잭션 관리를 통해 데이터의 일관성과 안정성을 보장한다.

- **파일 시스템 문제 해결:** 파일 시스템에서는 취약했던 보안 문제, 동시 접근 시 데이터가 꼬이는 문제 등을 해결해준다.

4. **데이터 중복 최소화 및 일관성 유지**
   - **역할:** DBMS는 정규화(Normalization)라는 과정을 통해 데이터를 여러 테이블로 분리하여 저장함으로써 불필요한 데이터 중복을 줄인다. 데이터 중복이 줄어들면, 데이터를 수정할 때 여러 곳을 고칠 필요가 없어져 데이터 불일치 가능성이 현저히 낮아지고 저장 공간도 효율적으로 사용할 수 있다.
   - **예시:** 고객의 주소 정보는 `고객` 테이블에 한 번만 저장하고, `주문` 테이블에서는 해당 고객의 ID만 참조하도록 하면, 고객이 이사했을 때 `고객` 테이블의 주소만 수정하면 모든 주문에서 변경된 주소를 참조할 수 있다.
   - **파일 시스템 문제 해결:** 파일에 데이터를 저장할 때 흔히 발생했던 데이터 중복 및 그로 인한 데이터 불일치 문제를 구조적으로 해결한다.

5. **데이터 백업 및 복구 (Data Backup and Recovery)**
   - **역할:** DBMS는 시스템 장애(예: 하드디스크 고장, 정전)나 사용자 실수로 인해 데이터가 손상되거나 유실될 경우를 대비하여, 데이터를 주기적으로 백업하고 문제 발생 시 안전하게 복구할 수 있는 기능을 제공한다. 로그(Log) 파일을 기록하여 변경 사항을 추적하고, 이를 통해 특정 시점으로 데이터를 복원할 수도 있다.
   - **파일 시스템 문제 해결:** 파일 시스템에서는 수동으로 백업해야 하고 복잡했던 데이터 복구 문제를 훨씬 체계적이고 안정적으로 처리해준다.

정리하자면, DBMS는 데이터를 단순히 저장하는 것을 넘어, **데이터를 정의하고, 조작하고, 관리함으로써 데이터의 품질, 보안, 안정성을 보장하는 아주 중요한 소프트웨어**다.

우리가 앞으로 다룰 MySQL도 바로 이 DBMS의 한 종류다. DBMS가 어떤 원리로 데이터를 관리하고 어떤 기능을 제공하는지 이해해야, 우리가 왜 MySQL을 배우고 어떻게 활용해야 하는지 그 본질을 알 수 있다. 그냥 "MySQL은 데이

터를 저장하는 프로그램" 정도로만 알고 넘어가면, 그 강력한 기능을 제대로 써먹을 수 없다.

> **데이터베이스 실무 용어**
> 보통 실무에서는 데이터베이스와 DBMS를 엄밀하게 구분하지는 않고, 둘을 합쳐서 데이터베이스(DB)라고 통칭해서 말한다. 그래서 DB라고 하면 DBMS와 그 DBMS가 관리하는 DB를 모두를 합친 것으로 이해하면 된다.
> 강의에서도 둘을 엄밀하게 구분하지 않고, DB로 통칭해서 사용하겠다.

관계형 데이터베이스 vs NoSQL

데이터베이스의 역사를 잠깐 되짚어 보자. 처음에는 단순 파일 시스템으로 데이터를 저장했고, 파일 시스템의 문제를 해결하기 위해 이어서 계층형과 네트워크형 DBMS가 나타났다. 이후 오늘날 가장 널리 쓰이는 **관계형 DBMS(RDBMS)**가 자리 잡았으며, 최근에는 빅데이터 처리나 특수 요구 사항에 대응하기 위해 **NoSQL DBMS**가 빠르게 발전하고 있다.
참고로 계층형과 네트워크형 DBMS는 시장에서 사장된 지 오래 되었고, 더는 사용되지 않는다.

이 모든 변화의 핵심 질문은 항상 같다. **"데이터를 어떻게 하면 더 효과적이고 더 안전하게 관리할 수 있을까?"**

DBMS는 목적과 특성에 따라 여러 종류가 있지만, 최근에는 크게 두 분류로 구분된다.

- **관계형 데이터베이스 관리 시스템(RDBMS)**
  - 현재 가장 널리 사용되는 데이터베이스 시스템
  - 테이블 간의 관계를 기반으로 데이터를 구조화
- **NoSQL(Not Only SQL) 데이터베이스 관리 시스템**
  - NoSQL은 "Not Only SQL"의 줄임말
  - 빅데이터와 특정 요구사항에 맞춰 최근 등장한 시스템
  - 관계형 데이터베이스의 한계를 보완

관계형 데이터베이스 관리 시스템 (RDBMS - Relational Database Management System)

RDBMS는 줄여서 보통 **RDB**라고 부른다. 워낙 많이 사용되기 때문에 통상적으로 DB라고 하면 바로 이 RDBMS를 뜻하기도 한다.

이것이 바로 **우리가 집중적으로 배울 대상**이다. RDBMS는 데이터를 **테이블(Table)**이라는 정형화된 구조에 저장한다. 테이블은 행(Row, 레코드라고도 함)과 열(Column, 필드 또는 속성이라고도 함)로 구성되며, 마치 엑셀 시트와 비슷한 모습을 떠올리면 된다. 그리고 이 테이블들 간의 관계를 통해 데이터를 연결하고 관리한다.

고객 (customers) 테이블

customer_id (PK)	first_name	last_name	email	phone_number
1	John	Doe	john.doe@email.com	010-1234-5678
2	Jane	Smith	jane.smith@email.com	010-1234-0001
3	Alice	Johnson	alice.j@email.com	010-1234-0002
4	Bob	Brown	b.brown@email.com	010-1234-0003

주문 (orders) 테이블

order_id (PK)	customer_id (FK)	order_date	total_amount
101	1	2025-03-15	55000
102	2	2025-03-17	120000
103	1	2025-03-20	30000
104	3	2025-03-22	75000
105	4	2025-03-25	25000
106	2	2025-03-28	89000

고객과 주문 두 테이블은 `customer_id`를 통해 관계를 맺고 있다. 즉, 주문 테이블의 `customer_id`를 보면 어떤 고객이 해당 주문을 했는지 고객 테이블의 `customer_id`를 통해 찾아낼 수 있다.

대표적인 RDBMS 종류

각각의 장단점과 자세한 내용은 뒤에서 설명한다. 지금은 이렇게 다양한 RDBMS 소프트웨어 제품이 있다는 것만 알아두자.

- **MySQL**(우리가 배울 것!)
- MariaDB(MySQL에서 파생)
- Oracle

- Microsoft SQL Server
- PostgreSQL

왜 RDBMS가 기본인가?

- **정형화된 데이터 관리:** 데이터 구조가 명확하고, 데이터 타입과 제약조건을 통해 데이터의 무결성을 강력하게 보장한다. 우리 쇼핑몰의 고객 정보, 상품 정보, 주문 내역처럼 구조가 명확하고 일관성이 매우 중요한 데이터에 적합하다.
- **ACID 트랜잭션 보장:** 위에서 언급한 트랜잭션의 ACID (원자성, 일관성, 고립성, 지속성) 특성을 잘 지원하여 데이터 처리의 신뢰성이 매우 높다. 돈과 관련된 금융 거래나 쇼핑몰의 주문/결제 처리에 필수적이다.
- **SQL이라는 표준 질의어:** SQL(Structured Query Language)이라는 강력하고 표준화된 언어를 사용하여 복잡한 데이터 조회나 조작을 쉽게 할 수 있다.
- **성숙한 기술과 풍부한 생태계:** 수십 년간 발전해 온 기술로 매우 안정적이고, 관련 문서, 커뮤니티, 도구들이 풍부하여 문제 해결이나 학습이 용이하다.
- **실무에서의 압도적인 사용 빈도:** 백엔드 개발자가 된다면, **실무에서 거의 대부분은 RDBMS를 기본으로 사용하게 될 것이다.** 다른 종류의 DB는 특정 필요에 따라 옵션으로 사용된다. RDBMS 없이 프로젝트를 진행하는 경우는 (아주 특별한 경우가 아니라면) 거의 없다고 봐도 무방하다.

그럼 NoSQL과 같은 다른 DB는 언제 배우면 좋을까? **먼저 RDBMS를 제대로 마스터하고 나서 필요할 때 배우면 된다.** 참고로, 데이터를 관리하는 대부분의 문제는 RDBMS로 충분히 해결할 수 있다.

NoSQL 데이터베이스 관리 시스템 (Not Only SQL)

NoSQL은 "Not Only SQL"의 약자로, 관계형 모델을 사용하지 않거나, SQL을 주요 데이터 접근 언어로 사용하지 않는 DBMS들을 통칭한다. RDBMS가 모든 상황에 최적인 것은 아니기 때문에, 특정 요구사항(예: 엄청나게 많은 비정형 데이터 처리, 매우 빠른 읽기/쓰기 속도, 유연한 데이터 모델 등)을 만족시키기 위해 등장했다.

대표적인 NoSQL을 간단히 살펴보자.

NoSQL 4대 분류 – 간단 정리

분류	데이터 단위	핵심 장점	대표 사례	대표 DBMS
키-값 저장소	키 → 값 한 쌍	초고속 단건 읽기/쓰기	세션, 캐시	Redis, Memcached
문서 DB	JSON/BSON 문서	유연한 스키마	상품 카탈로그	MongoDB, Couchbase

컬럼 패밀리 저장소	열(Column) 그룹	대용량 분산/분석	로그/시계열	Cassandra, HBase
그래프 DB	노드, 엣지	복잡한 관계 탐색	추천/이상 탐지	Neo4j

NoSQL은 RDBMS를 대체한다기보다는 **상호 보완적인 관계**로 이해하는 것이 좋다. 우리 쇼핑몰에서도 핵심 데이터는 RDBMS에 저장하되, 특정 기능(예: 실시간 인기 검색어, 사용자 세션 관리, 상품 추천 등)에는 NoSQL을 함께 사용할 수도 있다.

실무 이야기: NoSQL 도입

- RDB는 실무에 필요한 대부분의 문제를 해결할 수 있는 만능 해결사이다. 따라서 서비스가 작을 때는 보통 RDB만 사용하다가 서비스가 점점 커지고, 사용자가 늘어나면서 NoSQL이 꼭 필요한 상황들이 발생하면 그때 각 상황에 맞는 NoSQL을 부분적으로 도입하는 것이 좋다.
- 각각의 DB가 모두 깊이가 있다. RDB 하나만 제대로 이해하기도 쉽지 않다. 처음부터 무리하게 다양한 NoSQL을 함께 사용하면 관리해야 하는 시스템이 점점 늘어나므로 시스템 운영이 매우 어려워진다.
- 예를 들어보자. 실무에서는 고객이 요청할 때 마다 이 고객이 정상적인 사용자인지 아닌지 확인이 필요하다. 쉽게 이야기해서 고객이 요청할 때 마다 데이터베이스를 꼭 확인해야 한다. 이때 RDB를 사용하는 것 보다 Redis 같은 키-값 저장소를 사용하는 것이 더 빠르고 효율적이다. 그래서 처음부터 Redis를 도입하는 경우도 있다. 하지만 Redis도 결국 또 하나의 시스템이므로 관리 포인트가 늘어난다는 점을 매우 신중하게!!!!! 고려해야 한다. 참고로 사용자가 수백만 명 이상이 되더라도 RDB만으로도 충분히 대부분의 서비스를 운영할 수 있다.

정리 - 데이터베이스 관리 시스템 종류

- 계층형 (사라짐)
- 네트워크형 (사라짐)
- 관계형 데이터베이스 (메인)
- NoSQL (옵션)
  - 키-값 저장소
  - 문서 DB
  - 컬럼 패밀리 저장소
  - 그래프 DB

이렇게 다양한 DBMS가 있지만, 다시 한번 강조한다. **우리의 첫 번째 목표는 RDBMS, 그중에서도 MySQL을 확실하게 마스터하는 것이다!** 이것만 잘해도 대부분의 상황에 잘 대처할 수 있다.

관계형 데이터베이스 종류

다양한 RDBMS 비교

실무에서 주로 사용되는 다양한 RDBMS들을 간단히 알아보자.

Oracle (오라클 데이터베이스)

- 매우 강력하고, 기능이 정말 많고, 안정성도 최고 수준인 상용 RDBMS의 대표 주자다. 주로 대기업이나 금융권처럼 아주 높은 수준의 성능과 안정성, 그리고 복잡한 기능을 필요로 하는 곳에서 많이 사용된다.
- **단점:** 라이선스 비용이 매우 비싸고, 기능이 많은 만큼 배우고 관리하는 데 더 많은 전문성이 요구된다.

MySQL (마이에스큐엘)

- 전 세계적으로 **가장 널리 사용되는 오픈 소스 RDBMS** 중 하나이다. 특히 웹 애플리케이션 개발 분야에서 압도적인 점유율을 자랑하며, 빠른 속도와 사용 편의성, 그리고 방대한 커뮤니티와 자료 덕분에 초보자부터 전문가까지 폭넓게 사용된다.
- MySQL 커뮤니티 에디션은 무료로 다운로드해서 사용할 수 있다. 상용 버전(Enterprise Edition)에는 추가 기능이나 기술 지원 서비스가 포함되어 유료이지만, 대부분의 경우 커뮤니티 에디션만으로도 충분히 강력한 기능을 활용할 수 있다.
- 배우기 쉽고 사용이 간편하며, 다양한 운영체제에서 잘 작동한다. 풍부한 문서와 커뮤니티 지원을 받을 수 있으며, 특히 중소규모 서비스나 스타트업에서 비용 효율적으로 데이터베이스를 구축하고 운영하기에 적합하다. 최근에는 대기업이나 금융 서비스에도 많이 사용하는 추세이다.
- **단점:** 복잡한 분석 쿼리나 대규모 트랜잭션 처리 성능은 상용 RDBMS나 PostgreSQL에 비해 다소 부족하다는 평가도 있었으나, 지속적인 성능 개선과 새로운 기능 추가로 많은 부분에서 격차를 줄여나가고 있다.

PostgreSQL (포스트그레스큐엘, 줄여서 포스트그레스)

- MySQL과 마찬가지로 매우 훌륭한 오픈 소스 RDBMS다. 기능적으로는 MySQL보다 더 뛰어나다고 평가받는 부분이 많다 (예: 복잡한 쿼리 처리 능력, 표준 SQL 준수, 확장 기능 등). 최근 몇 년간 인기가 급상승하며 많은 곳에서 사용되고 있다.
- **단점:** 아직까지 국내 사용자가 많지 않다. 그래도 점점 늘어가는 추세이다.

Microsoft SQL Server (MS SQL)

- 마이크로소프트에서 만든 RDBMS로, 특히 윈도우 서버 환경이나 닷넷(.NET) 기반 애플리케이션과 궁합이 좋다. 강력한 성능과 다양한 도구를 제공한다.
- **단점:** 주로 마이크로소프트 기술 스택을 사용하는 환경에서 선택되며, 오픈 소스 진영과는 약간 거리가 있다.

MariaDB (마리아DB)

- MySQL의 "쌍둥이 동생" 이라고 생각하면 된다. MySQL이 오라클에 인수된 후, MySQL의 창립 멤버들이 "오픈 소스 정신을 계속 이어가겠다"며 만든 MySQL 기반의 RDBMS다. MySQL과 거의 대부분 호환되기 때문에 MySQL 대신 MariaDB를 사용해도 대부분 문제가 없다.
- 참고로 우리가 MySQL을 배운다는 것은 MariaDB를 이해하는 것과 거의 같다고 봐도 무방하다.

H2 Database (또는 SQLite)

- 주로 **임베디드(내장형) 데이터베이스**로 사용된다. 즉, 별도의 서버를 띄우는 것이 아니라 애플리케이션 자체에 포함되어 실행되는 가벼운 데이터베이스다. 주로 개발 단계에서의 빠른 테스트, 모바일 앱 내부 데이터 저장, 소규모 데스크톱 애플리케이션 등에 적합하다.

MySQL 소개

MySQL은 스웨덴의 한 회사(MySQL AB)에서 처음 개발되었고, 이후 썬 마이크로시스템즈(Sun Microsystems)를 거쳐 현재는 오라클(Oracle) 회사에 인수되어 관리되고 있다. 하지만 여전히 오픈 소스 라이선스를 따르기 때문에 많은 사용자들이 부담 없이 접근할 수 있다. 참고로, MySQL의 초기 개발자들이 오라클의 인수 이후 독립적으로 만든 MariaDB라는 것도 있는데, MySQL과 거의 완벽하게 호환되면서 독자적인 발전을 이어가고 있다. 이것도 아주 훌륭한 RDBMS다.

MySQL은 처음부터 **빠른 속도, 높은 안정성, 그리고 사용 편의성**을 목표로 설계되었다. 특히 웹 애플리케이션과의 궁합이 좋아서 수많은 웹사이트와 온라인 서비스의 백엔드 데이터베이스로 사랑받아 왔다.
MySQL은 전 세계적으로 가장 널리 사용되는 **오픈 소스(Open Source) 관계형 데이터베이스 관리 시스템 (RDBMS)** 중 하나다.

RDBMS의 호환성과 ANSI SQL 표준

"관계형 데이터베이스마다 사용법이 서로 달라서 하나를 배워도 다른 건 새로 배워야 하는 거 아닌가?" 라고 걱정할 수 있다. 하지만 그렇지 않다. 바로 **ANSI/ISO**에서 지정한 SQL 표준 덕분이다. 여기서 RDBMS들이 공통적으로 사용할 수 있는 표준 SQL 문법을 정의했다. Oracle, MySQL, MS SQL, PostgreSQL 등 대부분의 RDBMS는 이 표준을 준수하려고 노력한다.
물론, 각 RDBMS는 표준 SQL 외에도 자신만의 고유한 추가 기능(함수나 문법)을 가지고 있다. 하지만 데이터를 다루는 가장 핵심적인 부분인 조회(SELECT), 추가(INSERT), 수정(UPDATE), 삭제(DELETE) 등의 기본 문법은 대부분 SQL 표준을 따르기 때문에 거의 모든 RDBMS를 비슷하게 다룰 수 있다.
결론적으로, MySQL을 통해 RDBMS의 핵심 개념과 표준 SQL을 제대로 배우면, 다른 종류의 데이터베이스(Oracle, PostgreSQL 등)를 접하더라도 쉽게 적응하고 사용할 수 있다. 쉽게 이야기하자면 자동차 운전을 배우면 다른 차종도 금방 운전할 수 있는 것과 같은 이치다.

실무 이야기 - DB 선택

실무에서 현실적으로 가능한 선택지는 다음과 같은 이유로 오라클이나 MySQL(MariaDB 포함) 두 가지 정도이다.

1. **인력 수급의 현실**
    - 오라클과 MySQL은 국내에서 관련 인력을 구하기가 상대적으로 쉽다. 개발자나 DBA(데이터베이스 관리자)를 채용할 때도 이 두 기술에 대한 경험자를 찾기가 훨씬 용이하다.
    - 반면 PostgreSQL이나 다른 RDBMS 전문가는 찾기 어려워서, 기술적으로 우수하더라도 현실적인 선택에서 제외되는 경우가 많다.
2. **비용과 규모에 따른 선택**
    - **대기업/금융권/공공기관**: 오라클을 선택하는 경우가 많다. 높은 라이선스 비용을 감당할 수 있고, 미션 크리티컬한 시스템에서 요구되는 고급 기능과 기술 지원이 필요하기 때문이다.
    - **중소기업/스타트업/웹서비스**: MySQL을 선택하는 경우가 압도적으로 많다. 무료로 사용할 수 있고, 웹 애플리케이션 개발에 최적화되어 있으며, 빠른 개발과 운영이 가능하다.
3. **학습과 개발 편의성**
    - MySQL은 상대적으로 배우기 쉽고, 한글 자료도 풍부하다.
    - 오픈 소스 특성상 다양한 도구와 라이브러리가 잘 개발되어 있어서 개발 생산성이 높다.
4. **최근 동향**
    - **비용 절감 니즈:** 오라클의 높은 라이선스 및 유지보수 비용에 대한 부담으로 인해 대기업, 금융권, 공공기관에서도 적극적으로 오픈 소스 RDBMS(MySQL, MariaDB, PostgreSQL 등) 도입을 검토하거나 이미 주요 시스템에 적용하는 사례가 크게 늘고 있다.
    - **오픈 소스 성능 및 기능 향상:** 과거에는 오픈 소스 RDBMS가 대규모 트랜잭션 처리나 안정성, 보안 기능 면에서 상용 RDBMS에 미치지 못한다는 인식이 있었으나, 지속적인 기술 발전으로 MySQL, MariaDB, PostgreSQL 같은 경우 충분히 기능을 제공하며 성능 또한 크게 향상되었다.

결론적으로, IT 업계에서는 "안정성을 원하면 오라클, 비용 효율성과 빠른 개발을 원하면 MySQL"이라는 공식이 자리 잡혀 있다. 물론 최근에는 PostgreSQL이나 NoSQL 데이터베이스들도 점점 더 많이 사용되고 있지만, 여전히 오라클과 MySQL이 주류를 이루고 있다. 참고로 MariaDB의 경우 MySQL과 거의 같기 때문에 어렵지 않게 선택할 수 있다. 이런 이유로 우리도 MySQL을 중심으로 RDBMS를 학습하게 될 것이다. MySQL을 제대로 이해하면 SQL 표준 덕분에 다른 RDBMS를 배우는 것도 훨씬 수월해진다.

강의 목표

실습은 **MySQL**을 중심으로 진행되지만, 강의 내용은 **MySQL뿐만 아니라 RDBMS 전체를 아우르는 보편적인 지식**을 전달하는 데 초점을 맞출 것이다. 그래서 특정 데이터베이스에 얽매이지 않는 폭넓은 이해를 목표로 한다.

비유를 하자면 이 강의에서 우리는 **MySQL**이라는 자동차를 운전하는 법을 배울 것이다. 하지만 단순히 특정 모델의 자동차 조작법만 익히는 것이 아니라, 액셀, 브레이크, 핸들처럼 **어떤 자동차(RDBMS)를 타든 기본적으로 알아야 할 운**

전 원리(RDBMS의 핵심 개념)를 배우는 데 중점을 둘 것이다. 그래서 다른 RDBMS를 다룰 때도 큰 도움이 될 것이다.

관계형 데이터베이스 핵심 개념

관계형 모델의 핵심 개념: 모든 데이터는 '표'에서 시작한다.

관계형 데이터베이스는 데이터를 '표'의 형태로 관리한다. 이것은 사실 우리가 이미 엑셀을 통해 익숙하게 봐왔던 구조다. 용어만 다를 뿐, 개념은 비슷하다.

- **테이블 (Table)**: 관계형 데이터베이스에서 데이터를 저장하는 가장 기본적인 구조다. 엑셀의 '시트(Sheet)'와 거의 동일한 개념이다. 우리는 쇼핑몰 데이터를 관리하기 위해 `고객 테이블`, `상품 테이블`, `주문 테이블` 등을 만들게 될 것이다. 이 테이블은 특정 주제와 관련된 데이터들의 집합이다.

- **행 (Row)**: 테이블의 각 가로줄을 의미한다. 하나의 행은 개별적인 데이터 항목 하나를 나타낸다. 예를 들어, `고객 테이블`에서 하나의 행은 고객 한 명의 정보를 의미한다. '이순신' 고객의 정보, '강감찬' 고객의 정보 등이 각각 하나의 행이 된다.
  - **실무 용어**: 행은 '레코드(Record)' 또는 '튜플(Tuple)'이라고도 불린다.

- **열 (Column)**: 테이블의 각 세로줄을 의미한다. 열은 테이블에 어떤 종류의 데이터가 저장될지를 정의한다. `고객 테이블`이라면 `고객번호`, `이름`, `연락처`, `주소` 등이 각각의 열이 된다.
  - **실무 용어**: 열은 '속성(Attribute)' 또는 '필드(Field)'라고도 한다.

아래 `고객 테이블` 예시를 통해 다시 한번 정리해 보자.

[customers 테이블]

customer_id	name	phone	address
1	이순신	010-1111-1111	서울시 강남구
2	강감찬	010-2222-2222	서울시 서초구
3	세종대왕	010-3333-3333	서울시 종로구

- 이 전체 표가 바로 **테이블**이다. (`customers`)

- '이순신' 고객의 데이터 한 줄 (`1, 이순신, 010-1111-1111, 서울시 강남구`)이 바로 **행(Row)**이다. 이 테이블에는 총 3개의 행이 있다.
- `customer_id`, `name`, `phone`, `address` 와 같은 세로 항목들이 바로 **열(Column)**이다. 이 테이블은 4개의 열로 구성되어 있다.

관계형 데이터베이스의 시작은 이렇게 우리가 이미 알고 있는 '표'의 개념을 그대로 가져오는 것에서부터 출발한다. 다만, 데이터베이스는 여기에 몇 가지 강력한 규칙을 추가하여 엑셀의 한계를 뛰어넘는다. 이제 그 핵심 규칙들을 하나씩 배워보자.

핵심 개념 1: 기본 키(Primary Key) - 수많은 데이터 속에서 '단 하나'를 식별하는 방법

문제 상황: 어떻게 데이터를 유일하게 구분할까?
쇼핑몰을 운영하다 보니, 이름도, 주소도, 심지어 주문한 상품까지 완전히 똑같은 주문이 두 건 들어왔다고 가정해 보자. 동명이인이 같은 날 같은 상품을 주문한 것이다.

[orders 테이블 (나쁜 예시)]

주문일자	고객이름	고객주소	상품명
2025-06-12	네이트	서울시 마포구	A청바지
2025-06-12	네이트	서울시 마포구	A청바지

이때, 고객 한 명이 주문을 취소해달라고 연락했다. 우리는 이 두 주문 중 어떤 것을 취소해야 할까? 이름, 주소, 상품명 그 어떤 정보로도 두 주문을 구분할 수가 없다. 이것이 바로 데이터의 '식별' 문제다.

해결: 각 데이터를 유일하게 만들어주는 식별자, 기본 키 (Primary Key)
이 문제를 해결하기 위해 관계형 데이터베이스는 **기본 키(Primary Key, 줄여서 PK)** 라는 개념을 도입했다.

기본 키란, 테이블에 있는 모든 행(Row)들 중에서 특정 행 하나를 유일하게 식별할 수 있는 열(Column) 또는 열들의 조합을 말한다.

기본 키는 두 가지 중요한 규칙을 반드시 지켜야 한다.

1. **고유성 (Uniqueness)**: 기본 키로 지정된 열의 값은 같은 테이블 내에서 절대 중복될 수 없다. 모든 행이 서로 다른 값을 가져야 한다.
2. **NOT NULL**: 기본 키로 지정된 열에는 반드시 값이 있어야 한다. 비어있거나(NULL) 값이 없는 상태는 허용되지

않는다.

> `NULL`은 값이 존재하지 않는다는 표현이다.

이 규칙을 적용하여 위의 `orders` 테이블을 다시 설계해 보자. `order_id` 라는 이름의 열을 추가하고, 이 열을 기본 키로 지정하는 것이다.

[orders 테이블 (좋은 예시)]

order_id (PK)	주문일자	고객이름	고객주소	상품명
1001	2025-06-12	네이트	서울시 마포구	A청바지
1002	2025-06-12	네이트	서울시 마포구	A청바지

이제 다른 모든 정보가 똑같더라도, 우리는 `order_id` 가 `1001`인 주문과 `1002`인 주문을 명확하게 구분할 수 있다. 고객이 주문을 취소해달라고 요청하면, 주문번호를 확인해서 정확한 처리가 가능해진다.

이처럼 기본 키는 각 데이터에 주민등록번호를 부여하는 것과 같다. 이름, 생일, 출생지가 모두 같은 사람이 존재할 수 있지만, 주민등록번호는 절대 중복되지 않는 것처럼 말이다.

왜 기본 키가 필수적인가?
수많은 데이터 속에서 특정 데이터 하나를 빠르고 정확하게 찾아내고, 수정하고, 삭제하기 위해서다. 기본 키가 없다면 우리는 데이터의 바다에서 원하는 정보를 특정할 수 없다. 따라서 모든 테이블에는 기본 키를 설정하는 것이 원칙이다.

> ✨ **실무 이야기**
> 실무에서는 보통 `id`라는 이름의 열을 만들고, 1부터 시작하여 데이터가 추가될 때마다 1씩 자동으로 증가하는 정수(Integer) 값을 기본 키로 사용하는 경우가 가장 흔하다. 고객 테이블은 `customer_id`, 상품 테이블은 `product_id` 와 같이 `테이블명_id` 형식으로 이름을 짓는 것이 일반적인 관례다.

핵심 개념 2: 외래 키(Foreign Key) - 따로 떨어진 표들을 '관계'로 묶는 방법

우리는 이제 데이터를 주제별로 나누어 테이블로 만들고, 각 테이블의 데이터는 기본 키로 식별할 수 있게 되었다. 이제

엑셀의 문제였던 '데이터 중복'과 '데이터 불일치' 문제를 해결할 준비가 거의 끝났다.

문제 상황: 쪼개진 표들을 어떻게 연결할까?
데이터 중복을 피하기 위해, 우리는 정보를 '고객'과 '주문'이라는 두 개의 테이블로 나누어 저장하기로 했다.

[customers 테이블]

customer_id (PK)	name	phone	address
1	이순신	010-1111-1111	서울시 강남구
2	강감찬	010-2222-2222	서울시 서초구

[orders 테이블]

order_id (PK)	order_date	item_name
1001	2025-06-11	A청바지
1002	2025-06-11	B티셔츠
1003	2025-06-12	C모자

이제 고객 정보는 `customers` 테이블에 한 번만 저장되므로, 이순신 고객이 100번을 주문해도 그의 정보는 중복해서 저장되지 않는다. 주소가 바뀌면 `customers` 테이블의 해당 행 딱 한 곳만 수정하면 되므로 데이터 불일치 문제도 해결된다.

그런데 여기서 새로운 질문이 생긴다.
"1001번 주문을 한 고객이 누구인지 어떻게 알 수 있는가?"
두 테이블이 완전히 따로 떨어져 있기 때문에, 어떤 주문이 어떤 고객에 의해 이루어졌는지 연결할 방법이 보이지 않는다.

해결: 관계의 고리, 외래 키 (Foreign Key)
이 문제를 해결하는 것이 바로 관계형 데이터베이스의 핵심이자 꽃이라 불리는 **외래 키(Foreign Key, 줄여서 FK)**다.
아이디어는 놀랍도록 간단하다.

외래 키란, 한 테이블(A)의 열(Column)이 다른 테이블(B)의 기본 키(Primary Key) 값을 참조하는 것을 말한다.

즉, `orders` 테이블에 '이 주문은 어떤 고객의 것인가?'를 알려주는 정보를 추가하면 된다. 이때 `customers` 테이블의 기본 키인 `customer_id`를 `orders` 테이블에 가져와 사용하는 것이다.

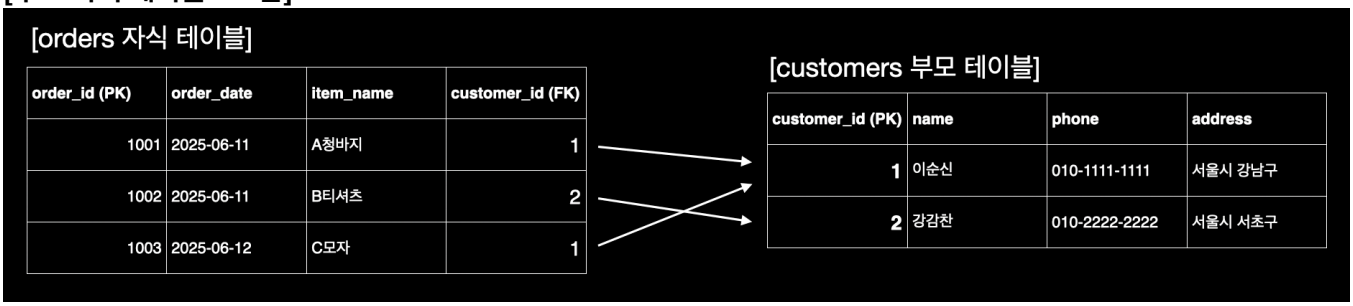
[customers 테이블] (참조 **당하는** 쪽, 부모 테이블)

customer_id (PK)	name	phone	address
1	이순신	010-1111-1111	서울시 강남구
2	강감찬	010-2222-2222	서울시 서초구

[orders 테이블] (참조 **하는** 쪽, 자식 테이블)

order_id (PK)	order_date	item_name	customer_id (FK)
1001	2025-06-11	A청바지	1
1002	2025-06-11	B티셔츠	2
1003	2025-06-12	C모자	1

[부모 자식 테이블 - 그림]



이제 `orders` 테이블을 보자. `1001`번 주문의 `customer_id`는 `1`이다. 이 `1`이라는 값을 가지고 `customers` 테이블로 가보면, `customer_id`가 `1`인 고객이 '이순신'이라는 것을 바로 알 수 있다.

정리하면 다음과 같다.

- `1001`번 주문의 `customer_id`가 `1`이므로 '이순신' 고객의 주문임을 알 수 있다.
- `1002`번 주문의 `customer_id`가 `2`이므로 '강감찬' 고객의 주문임을 알 수 있다.
- `1003`번 주문의 `customer_id`가 `1`이므로 '이순신' 고객의 주문임을 알 수 있다.

이처럼 다른 테이블의 기본 키를 참조하는 `orders` 테이블의 `customer_id` 열, 이것이 바로 **외래 키**다. 외래 키는 두 테이블을 연결하는 '관계의 고리' 역할을 한다. 별것 아닌 것 같지만, 관계형 데이터베이스의 모든 강력함이 바로 이

지점에서 나온다.

부모와 자식의 관계

- 두 테이블이 FK 값을 통해 관계가 있을 때 한쪽을 부모, 한쪽을 자식이라 한다.
- 자식 테이블은 FK 값을 통해 부모 테이블을 참조한다. FK 값을 가진 곳이 자식 테이블이다.
- 여기서는 `orders` 테이블이 FK 값인 `customer_id`를 통해 `customers` 테이블을 참조한다.
  - 따라서 둘의 관계에서 `orders`가 자식 테이블이고, `customers`가 부모 테이블이 된다.

외래 키의 중요한 규칙

- **참조 무결성(Referential Integrity)**: 외래 키 열에 있는 값은, 반드시 부모 테이블(참조 당하는 쪽, 여기서는 `customers`)의 기본 키 값 중 하나이거나, 혹은 비어있어야(NULL) 한다. 예를 들어, `orders` 테이블의 `customer_id`에 `customers` 테이블에 존재하지도 않는 `99` 같은 값을 넣으려고 하면 데이터베이스가 "그런 고객은 존재하지 않습니다!"라며 오류를 발생시켜 막아준다. 이 덕분에 데이터의 정합성이 보장되는 것이다.

왜 외래 키를 사용해 테이블을 연결하는가?

- 데이터의 중복을 막고, 데이터의 일관성을 유지하며, 논리적으로 분리된 데이터들 사이에 '관계'를 맺어주기 위해서다. 이를 통해 우리는 작고 관리하기 쉬운 여러 개의 테이블로 전체 시스템을 구조화할 수 있다.

이것으로 관계형 데이터베이스의 가장 핵심적인 개념 세 가지, **테이블(Table), 기본 키(Primary Key), 외래 키(Foreign Key)**에 대해 알아보았다.

다음 시간부터는 본격적으로 데이터베이스를 사용해보자.

정리

데이터와 정보

- 데이터는 가공되지 않은 개별적인 사실이나 값이며, 그 자체로는 의미를 갖기 어렵다.
- 구조화된 데이터는 데이터에 꼬리표(Label)를 붙여 의미를 부여한 상태이지만, 통찰을 주지는 못한다.
- 정보는 구조화된 데이터를 특정 목적을 가지고 분석, 가공하여 얻어낸 유의미한 결과물이다.
- 데이터를 정보로 만드는 과정은 의사결정에 필수적이며, 데이터베이스 관리 시스템은 이 과정을 돕는 핵심 도구다.

데이터베이스 관리 시스템이 필요한 이유

- 컴퓨터 파일 시스템(텍스트 파일, 엑셀)으로 데이터를 관리하는 것은 초기에는 가능하지만, 규모가 커지면 심각한

문제점을 드러낸다.

- **파일 시스템의 문제점**
  - **데이터 중복과 불일치**: 여러 파일에 동일 정보가 중복 저장되어 수정 시 일부만 변경되면 데이터가 어긋난다.
  - **데이터 접근의 어려움**: 여러 파일에 흩어진 데이터를 조합하여 원하는 정보를 얻기 매우 복잡하고 비효율적이다.
  - **무결성 제약조건 적용의 어려움**: 데이터 형식이나 값에 대한 규칙(예: 가격은 0보다 커야 함)을 강제하기 어렵다.
  - **동시성 제어 문제**: 여러 사용자가 동시에 데이터를 수정할 때 데이터 일관성이 깨질 수 있다.
  - **보안 문제**: 사용자별로 데이터 접근 권한을 세밀하게 제어하기 어렵다.
  - **회복 및 백업의 어려움**: 장애 발생 시 데이터를 일관된 상태로 완벽하게 복구하기가 매우 복잡하다.
- 이러한 파일 시스템의 문제들을 해결하기 위해 데이터베이스가 등장했다.

데이터베이스 관리 시스템 (DBMS) 소개

- DBMS는 사용자와 데이터베이스 사이에서 데이터를 생성, 정의, 공유, 사용, 관리하는 기능을 수행하는 소프트웨어 시스템이다.
- 개발자는 DBMS가 제공하는 표준화된 언어(SQL 등)로 데이터를 요청하고, DBMS가 파일 시스템을 직접 다루어 복잡한 처리를 대신 수행한다.
- **DBMS의 핵심 기능**
  - **데이터 정의 기능 (DDL)**: 데이터의 구조, 형식, 관계, 제약조건을 설정한다.
  - **데이터 조작 기능 (DML)**: 데이터를 등록(Insert), 조회(Select), 수정(Update), 삭제(Delete)한다.
  - **보안, 동시성 제어, 트랜잭션 관리 기능**: 보안, 동시성 제어, 트랜잭션 관리를 통해 데이터의 정확성과 안정성을 보장한다.
- DBMS는 데이터 중복을 최소화하고, 장애 발생 시 데이터를 안전하게 복구하는 기능도 제공한다.

관계형 데이터베이스 vs NoSQL

- **관계형 데이터베이스(RDBMS)**: 데이터를 정형화된 테이블 구조에 저장하고, 테이블 간의 관계를 통해 데이터를 관리한다. 데이터 무결성과 일관성이 중요할 때 사용하며, SQL이라는 표준 언어를 사용한다. (예: MySQL, Oracle)
- **NoSQL(Not Only SQL)**: 관계형 모델을 사용하지 않는 DBMS를 통칭하며, 비정형 데이터 처리, 빠른 속도, 유연한 구조가 필요할 때 사용한다. RDBMS를 대체하기보다 상호 보완적인 관계다. (예: Redis, MongoDB)
- 실무에서는 대부분 RDBMS를 기본으로 사용하며, 필요에 따라 NoSQL을 부분적으로 도입한다.

관계형 데이터베이스 종류

- **주요 RDBMS**: Oracle(대기업/금융권), Microsoft SQL Server(윈도우 환경), PostgreSQL(기능/확장성 우수), MySQL(웹 서비스, 스타트업에서 압도적 사용), MariaDB(MySQL 기반 오픈 소스)
- **MySQL**: 세계적으로 가장 널리 쓰이는 오픈 소스 RDBMS로, 배우기 쉽고 커뮤니티가 활성화되어 있다. 웹 애플

리케이션 개발에 특히 강점을 보인다.

- 대부분의 RDBMS는 **SQL 표준**을 준수하므로, 하나(MySQL 등)를 제대로 배우면 다른 RDBMS에도 쉽게 적응할 수 있다.

관계형 데이터베이스 핵심 개념

- **테이블(Table)**: 데이터를 저장하는 표 형태의 기본 구조로, 행과 열로 구성된다.
- **행(Row/Record)**: 테이블의 각 가로줄로, 개별 데이터 항목 하나를 나타낸다.
- **열(Column/Field)**: 테이블의 각 세로줄로, 데이터의 종류(속성)를 정의한다.
- **기본 키 (Primary Key, PK)**: 테이블의 모든 행을 유일하게 식별하는 값. **고유성**과 **NOT NULL** 규칙을 반드시 지켜야 한다. 데이터의 식별을 위해 모든 테이블에 PK를 설정하는 것이 원칙이다.
- **외래 키 (Foreign Key, FK)**: 한 테이블의 열이 다른 테이블의 기본 키를 참조하는 것. 테이블 간의 관계를 맺어주며 데이터 중복을 방지하고 **참조 무결성**을 보장한다. 이를 통해 논리적으로 분리된 데이터를 연결하여 관리할 수 있다.