5. SQL - 조회와 정렬

#0.강의/2.데이터베이스로드맵/1.입문

조회 실습 데이터 준비

모든 데이터 분석과 조회의 첫걸음은 조회할 데이터가 '존재'해야 한다는 것이다. 우리 쇼핑몰이 이제 막 문을 열었고, 고맙게도 몇몇 고객이 가입하고 상품도 등록되었다고 가정해 보자. 앞으로의 실습을 위해 이 데이터를 먼저 우리 데이터베이스에 넣어두어야 한다.

문제와 풀이에서 다른 데이터베이스를 사용했기 때문에 먼저 데이터베이스 부터 변경하자.

```
USE my_shop;
```

샘플 데이터 입력

> **SQL 소스 파일 참고**
> 강의 자료가 PDF 파일이라 복잡한 SQL 코드를 복사할 때 오류가 발생할 수 있다.
> 이 경우, 섹션 1. 강의 소개와 수업 자료 ➜ SQL 소스 파일을 다운로드해서 사용하자.

예제를 깔끔하게 실행하기 위해서 먼저 앞서 생성한 테이블을 초기화 하자.

```
SET FOREIGN_KEY_CHECKS = 0; -- 비활성화
```

```
truncate table products;
truncate table customers;
truncate table orders;
SET FOREIGN_KEY_CHECKS = 1; -- 활성화
```

다음 SQL 쿼리는 테이블에 실제 데이터를 삽입하는 `INSERT` 문이다.

1. 고객 데이터 삽입 (`customers`)

```
INSERT INTO customers (name, email, password, address, join_date) VALUES
('이순신', 'yisunsin@example.com', 'password123', '서울특별시 중구 세종대로',
'2023-05-01'),
('세종대왕', 'sejong@example.com', 'password456', '서울특별시 종로구 사직로',
'2024-05-01'),
('장영실', 'youngsil@example.com', 'password789', '부산광역시 동래구 복천동',
'2025-05-01');
```

2. 상품 데이터 삽입 (`products`)

```
INSERT INTO products (name, description, price, stock_quantity) VALUES
('갤럭시', '최신 AI 기능이 탑재된 고성능 스마트폰', 10000, 55),
('LG 그램', '초경량 디자인과 강력한 성능을 자랑하는 노트북', 20000, 35),
('아이폰', '직관적인 사용자 경험을 제공하는 스마트폰', 5000, 55),
('에어팟', '편리한 사용성의 무선 이어폰', 3000, 110),
('보급형 스마트폰', NULL, 5000, 100);
```

3. 주문 데이터 삽입 (`orders`)

```
INSERT INTO orders (customer_id, product_id, quantity) VALUES
(1, 1, 1), -- 이순신 고객이 갤럭시 1개 주문
(2, 2, 1), -- 세종대왕 고객이 LG 그램 1개 주문
(3, 3, 1), -- 장영실 고객이 아이폰 1개 주문
(1, 4, 2), -- 이순신 고객이 에어팟 2개 추가 주문
(2, 2, 1); -- 세종대왕 고객이 LG 그램 1개 주문(추가 주문)
```

이제 모든 준비가 끝났다. 이 데이터를 가지고 본격적으로 데이터 조회의 세계로 들어가 보자.

샘플 데이터 확인

다음 SQL을 실행해서 데이터가 잘 입력되었는지 확인하자.

```
SELECT * FROM customers;
SELECT * FROM products;
SELECT * FROM orders;
```

`SELECT * FROM customers;` 실행 결과

customer_id	name	email	password	address	join_date
1	이순신	yisunsin@example.com	password123	서울특별시 중구 세종대로	2023-05-0 00:00:00
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-0 00:00:00
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-0 00:00:00

- `customer_id`는 자동으로 1부터 증가한다.

`SELECT * FROM products;` 실행 결과

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
5	보급형 스마트폰	NULL	5000	100

- `product_id`는 자동으로 1부터 증가한다.

`SELECT * FROM orders;` 실행 결과

order_id	customer_id	product_id	quantity	order_date	status
1	1	1	1	(현재 시각)	주문접수
2	2	2	1	(현재 시각)	주문접수
3	3	3	1	(현재 시각)	주문접수
4	1	4	2	(현재 시각)	주문접수
5	2	2	1	(현재 시각)	주문접수

- `order_id`는 자동으로 1부터 증가한다
- `order_date`는 DEFAULT 설정에 따라 현재 시각이 기록되며, `status`는 DEFAULT 설정에 따라 '주문접수'로 표시된다.

SELECT - 조회

우리가 열심히 개발한 쇼핑몰이 드디어 성공적으로 문을 열었다. 고객들이 하나둘씩 가입하기 시작했고, 상품도 등록했다. 이제 막 사업이 시작되는 흥분되는 순간이다. 그때, 대표가 우리에게 이렇게 말한다.

"우리 쇼핑몰에 가입한 전체 고객 명단을 지금 당장 보고 싶네!"

사업을 운영한다면 당연히 우리 고객이 누구인지, 어떤 상품이 있는지, 주문은 얼마나 들어왔는지 눈으로 직접 확인하고 싶을 것이다.

데이터베이스는 데이터를 저장하는 창고다. 하지만 창고에 물건을 쌓아두기만 하면 아무 의미가 없다. 필요할 때 물건을 꺼내서 확인하고, 사용해야 가치가 있다. 바로 이 '데이터를 꺼내보는 행위'를 **조회**라고 한다. 그리고 이 조회를 할 때 사용하는 가장 기본적인 명령어가 바로 `SELECT`다.

가장 간단한 데이터 조회

데이터를 조회하는 가장 기본적이고 간단한 방법부터 알아보자. `SELECT`와 `FROM` 두 가지 키워드만 알면 된다.

- `SELECT`: 무엇을 가져올 것인가? (조회할 열, 즉 컬럼을 지정한다)
- `FROM`: 어디에서 가져올 것인가? (데이터가 들어있는 테이블을 지정한다)

가장 단순하게 `customers` 라는 테이블에 들어있는 **모든** 고객 정보를 **전부** 꺼내보고 싶다고 해보자. 이럴 때 '모든 열(컬럼)'을 의미하는 특별한 기호인 `*` (애스터리스크, Asterisk) 와일드카드를 사용한다.

`SELECT * FROM 테이블이름;` 이 구문은 "이 테이블에 있는 모든 열의 데이터를 다 보여줘" 라는 의미다.

이제 대표님의 첫 번째 요청을 해결해 보자. `customers` 테이블의 모든 고객 정보를 조회해 보겠다.

[실습] customers 테이블의 모든 고객 정보 조회하기

```
SELECT * FROM customers;
```

[실행 결과]

customer_id	name	email	password	address	join_date
1	이순신	yisunsin@example.com	password123	서울특별시 중구 세종대로	2023-05-0 00:00:00
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-0 00:00:00
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-0 00:00:00

`customers` 테이블에 우리가 앞서 `INSERT` 했던 모든 데이터가 조회되었다.
이렇게 `SELECT *` 는 테이블의 구조나 내용을 빠르게 확인할 때 유용하다.

왜 필요한 데이터만 골라서 봐야 할까?

`SELECT *` 로 전체 고객 명단을 뽑아 대표님께 보고했더니, 아주 만족스러워한다. 그런데 잠시 후, 대표가 다시 우리를 찾는다.

"아주 좋아! 그런데 고객 전체 주소나 비밀번호 같은 민감한 정보는 필요 없고, 지금은 고객들의 이름과 이메일 주소만 빠르게 보고 싶어! 다시 뽑아주게."

`SELECT *`는 분명 편리하지만, 항상 좋은 것만은 아니다. 만약 `customers` 테이블에 수백만 명의 고객 데이터가 들어있고, 열(Column)도 50개가 넘는다고 상상해 봐라. 그저 이름과 이메일만 확인하고 싶을 뿐인데, `SELECT *`를 사용하면 어떤 문제가 발생할까?

1. **성능 저하**: 불필요한 데이터까지 모두 읽어오느라 데이터베이스 시스템에 큰 부담을 준다. 조회 속도가 당연히 느려진다.
2. **가독성 저하**: 내가 보고 싶은 데이터는 '이름'과 '이메일' 뿐인데, 수십 개의 열이 함께 표시되니 한눈에 파악하기 어렵다.
3. **네트워크 트래픽 낭비**: 데이터베이스 서버에서 우리 컴퓨터(클라이언트)로 데이터를 전송할 때, 필요 없는 데이터까지 함께 보내므로 네트워크 자원을 낭비하게 된다.

그래서 실무에서는 `SELECT *` 사용을 최소화하고, 꼭 필요한 열만 명시적으로 지정해서 조회하는 습관을 들여야 한다.

> **참고**
> 우리는 학습을 목적으로 **MySQL 데이터베이스 서버**와 **MySQL 워크벤치** 같은 **클라이언트 도구**를 한 컴퓨터에 함께 설치했다.
> 일반적으로는 **데이터베이스 서버**를 별도의 컴퓨터에 설치하고, 사용자는 자신의 PC에서 **MySQL 워크벤치**와 같은 **클라이언트 접속 도구**를 이용해 서버에 접속한다.

특정 열만 선택하기

`SELECT *` 대신, `SELECT` 절에 우리가 직접 보고 싶은 열의 이름들을 콤마(`,`)로 구분하여 적어주면 된다.

```
SELECT 열이름1, 열이름2, ... FROM 테이블이름;
```

이 구문은 "이 테이블에서, 내가 지정한 이 열들만 골라서 보여줘" 라는 의미다.
이제 대표의 두 번째 요청을 해결해 보자.
`customers` 테이블에서 고객의 이름(`name`)과 이메일 주소(`email`)만 선택하여 조회하겠다.

[실습] customers 테이블에서 고객 이름과 이메일 주소만 선택하여 조회하기

```
SELECT name, email FROM customers;
```

[실행 결과]

name	email
이순신	yisunsin@example.com
세종대왕	sejong@example.com
장영실	youngsil@example.com

실행 결과를 보자. `customer_id`, `password` 같은 불필요한 정보는 사라지고, 정확히 우리가 원했던 `name`과 `email` 열만 깔끔하게 조회되었다. 더욱 보기 좋고 효율적이다.

실무에서는 어떻게 사용할까? [열 이름에 별칭(Alias) 붙이기: AS]

이제 보고서를 좀 더 '있어 보이게' 만들어 보자. `name`, `email` 같은 영어로 된 열 이름은 개발자에게는 익숙하지만, 대표나 다른 팀의 동료들에게 보고할 때는 직관적이지 않다. 이럴 때 열 이름에 '별칭(Alias)'을 붙여서 조회 결과의 헤더를 바꿀 수 있다. `AS` 키워드를 사용한다.

```
SELECT 열이름1 AS 별칭1, 열이름2 AS 별칭2, ... FROM 테이블이름;
```

먼저 실행해보자.

[실습] 고객 이름은 '고객명'으로, 이메일 주소는 '이메일'로 별칭을 붙여 조회하기

```
SELECT name AS 고객명, email AS 이메일 FROM customers;
```

[실행 결과]

고객명	이메일
이순신	yisunsin@example.com
세종대왕	sejong@example.com
장영실	youngsil@example.com

결과 테이블의 헤더(머리글)가 우리가 지정한 '고객명'과 '이메일'로 바뀐 것을 볼 수 있다. 이렇게 하면 별도의 편집 없이도 조회 결과를 바로 보고서에 활용할 수 있다.

왜 별칭을 사용할까?

- **보고서의 가독성 향상**: 조회 결과를 '고객명', '이메일'처럼 한글로 표시하면 누가 봐도 이해하기 쉬운 보고서가 된다.
- **열 이름의 충돌 방지**: 지금은 테이블 하나만 다루지만, 나중에 여러 테이블을 연결(JOIN)해서 조회할 때, 서로 다른 테이블에 같은 이름의 열이 존재할 수 있다. 이때 별칭을 사용해 각 열을 명확하게 구분할 수 있다. (조인은 나중에 배운다)

참고로 `AS` 키워드는 생략하고 `SELECT 열이름1 별칭1` 과 같이 사용할 수도 있지만, 쿼리를 명확하게 만들기 위해 `AS`를 써주는 것이 좋은 습관이다.

AS를 사용하지 않는 경우

```
SELECT name 고객명, email 이메일 FROM customers;
```

> **별칭 이름 짓는 룰**
> 별칭도 앞서 배운 테이블과 컬럼 이름 규칙을 따르면 된다.
> 별칭도 마찬가지로 백틱으로 감싸면 특수문자를 사용할 수 있다.

WHERE - 기본 검색

지금까지 우리는 테이블의 모든 '행(Row)'을 가져오되, 보고 싶은 '열(Column)'을 선택하는 방법을 배웠다. 하지만 이것만으로는 부족하다.

어느 날 오후, 고객 서비스(CS) 팀에서 급한 목소리로 우리에게 요청한다.
"고객 한 분이 비밀번호를 잊어버렸다고 문의가 왔어요! 이메일 주소가 `yisunsin@example.com` 인 고객의 정보를 지금 바로 찾아서 알려주세요!"

지금은 고객이 3명뿐이라 눈으로 찾아도 되지만, 고객이 수만 명이라면 어떨까? `SELECT * FROM customers;` 로 모든 고객을 조회한 다음, 스크롤을 내려가며 `yisunsin@example.com` 이메일을 가진 고객을 눈으로 찾는 것은 거의 불가능에 가깝다. 시간도 오래 걸리고, 실수할 가능성도 매우 높다.

이처럼 우리는 전체 데이터가 아닌, **'특정한 조건을 만족하는'** 데이터만 콕 집어서 보고 싶을 때가 훨씬 많다. 이럴 때 사용하는 것이 바로 `WHERE` 절이다.

조건문의 시작: WHERE 절과 비교 연산자

`WHERE` 절은 `FROM` 절 바로 뒤에 위치하며, 우리가 원하는 행(Row)만 걸러내는 필터 역할을 한다.

```
SELECT 열이름
  FROM 테이블이름
 WHERE 조건;
```

`WHERE` 절에는 '조건문'이 들어가는데, 이 조건문은 보통 '비교 연산자'를 사용하여 만들어진다. 가장 기본이 되는 비교 연산자들은 다음과 같다.

연산자	의미	예시
`=`	같다	`WHERE name = '이순신'`
`!=` 또는 `<>`	같지 않다	`WHERE name != '세종대왕'`
`>`	크다	`WHERE price > 10000`
`<`	작다	`WHERE stock_quantity < 50`
`>=`	크거나 같다	`WHERE price >= 10000`
`<=`	작거나 같다	`WHERE stock_quantity <= 50`

> SQL에서 문자열, 날짜 값은 작은따옴표(')로 감싸준다. 숫자는 그대로 사용한다.

이제 CS팀의 요청을 해결해 보자. `customers` 테이블에서 이메일(`email`)이 `yisunsin@example.com`과 같은(`=`) 고객을 찾아보겠다.

[실습] customers 테이블에서 이메일이 `yisunsin@example.com`인 고객 조회하기

```sql
SELECT * FROM customers WHERE email = 'yisunsin@example.com';
```

[실행 결과]

customer_id	name	email	password	address	join_date
1	이순신	yisunsin@example.com	password123	서울특별시 중구 세종대로	2023-05-01 00:00:00

수만 건의 데이터가 있었더라도, 데이터베이스는 이메일이 정확히 일치하는 단 한 명의 고객 정보만 순식간에 찾아서 보여줄 것이다.

다른 예시도 살펴보자. 이번엔 상품 테이블에서 가격이 10000원 이상인 상품만 조회해 보자.

[실습] products 테이블에서 가격(price)이 10000원 이상인 상품만 조회해보자

```sql
SELECT * FROM products WHERE price >= 10000;
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

이렇게 `WHERE`과 비교 연산자를 사용하면 원하는 데이터를 필터링할 수 있다.

더 복잡한 조건을 원한다면?

`WHERE` 절의 기본을 익히니 일이 훨씬 수월해졌다. 이번엔 상품 관리팀에서 새로운 요청이 들어온다.

"가격이 5000원 이상이면서, 재고가 50개 이상인 상품을 찾아주세요."

이번에는 조건이 하나가 아니다. '가격'과 '재고'라는 두 가지 조건을 **모두** 만족해야 하는 상황이다. 이처럼 여러 개의 조건을 조합해야 할 때는 어떻게 해야 할까?

여러 조건들을 조합하는 논리 연산자: AND, OR, NOT

두 개 이상의 조건을 결합하여 더 정교한 필터링을 하고 싶을 때, '논리 연산자'를 사용한다.

- `AND` : 양쪽의 조건이 **모두** 참(True)일 때 최종적으로 참이 된다. '그리고', '~이면서'의 의미.
- `OR` : 양쪽의 조건 중 **하나라도** 참(True)이면 최종적으로 참이 된다. '또는', '~이거나'의 의미.
- `NOT` : 주어진 조건을 **부정**한다. '아닌'의 의미. (뒤에서 알아볼 IN, LIKE, BETWEEN, IS NULL 등과 함께 사용된다.)

상품 관리팀의 요청을 해결하려면 `AND`를 사용해야 한다. "가격이 5000원 이상" 이라는 조건과 "재고가 50개 이상" 이라는 두 조건을 `AND`로 연결하면 된다.

[실습] 가격이 5000원 이상이면서, 재고가 50개 이상인 상품 조회하기 (AND)

```sql
SELECT * FROM products WHERE price >= 5000 AND stock_quantity >= 50;
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
5	보급형 스마트폰	NULL	5000	100

- 에어팟은 가격이 3000원이어서 제외되고, LG 그램은 재고가 35개 밖에 없으므로 제외되었다. 갤럭시, 아이폰, 그리고 새로 추가된 보급형 스마트폰이 두 조건을 모두 만족하여 조회된다.

이번에는 `OR`를 사용해 보자. "가격이 20000원 이거나, 재고가 100개 이상인 상품을 찾아보겠다."
둘 중 하나의 조건만 만족해도 된다.

[실습] 가격이 20000원이거나, 재고가 100개 이상인 상품 조회하기 (OR)

```
SELECT * FROM products WHERE price = 20000 OR stock_quantity >= 100;
```

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
5	보급형 스마트폰	NULL	5000	100

- 'LG 그램'은 `price = 20000` 조건을 만족해서 선택되었고, '에어팟'과 '보급형 스마트폰'은 `stock_quantity >= 100` 조건을 만족해서 선택되었다. `OR`는 이렇게 여러 조건 중 하나만 만족해도 결과에 포함된다.

마지막으로 같지 않다는 의미인 `!=`를 사용해 보자. 가격이 20000원이 아닌 제품을 찾는다.

[실습] 가격이 20000원이 아닌 상품 조회하기 (!=)

```
SELECT * FROM products WHERE price != 20000;
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55

3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
5	보급형 스마트폰	NULL	5000	100

- 가격이 20000원인 LG 그램을 제외한 모든 상품이 선택되었다.

WHERE - 편리한 조건 검색

편리한 조건 검색: BETWEEN, IN, LIKE, IS NULL

매번 `=`이나 `>` 같은 기본 연산자만 사용하면 쿼리가 길어지고 비효율적일 수 있다. SQL은 더 편리한 검색을 위해 여러 유용한 연산자들을 제공한다.

BETWEEN: 특정 범위에 있는 값 찾기

MD(상품기획자)가 우리에게 묻는다. **"가격이 5,000원에서 15,000원 사이인 상품들만 모아서 보여주세요."**

이 요청을 기존 방식으로 해결하려면 `WHERE price >= 5000 AND price <= 15000`을 사용하면 된다.

```
SELECT * FROM products WHERE price >= 5000 AND price <= 15000;
```

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
5	보급형 스마트폰	NULL	5000	100

`BETWEEN`을 사용하면 같은 문제를 더 간결하게 표현할 수 있다.

`BETWEEN a AND b` 구문은 'a와 b 사이의 값(a, b 포함)'을 찾아준다. `a`에는 최솟값, `b`에는 최댓값을 넣는다.

[실습] products 테이블에서 가격이 5000원 이상 15000원 이하인 상품 조회하기

```
SELECT * FROM products WHERE price BETWEEN 5000 AND 15000;
```

- 실행 결과는 이전과 같다.
- `>= ... AND <= ...` 보다 더 직관적이고 SQL이 이해하기 쉬워졌다.
- BETWEEN은 양 끝값을 포함한다.

NOT BETWEEN: 특정 범위를 제외한 값 찾기

`BETWEEN` 앞에 `NOT`을 붙이면 정확히 그 반대의 의미가 된다. 왜 이런 기능이 필요할까?

다시 MD가 찾아왔다. **"가격이 5,000원에서 15,000원 사이인 상품들은 특별 할인 행사 대상입니다. 이 상품들을 제외한 나머지 모든 상품 목록을 보여주세요."**

`WHERE price < 5000 OR price > 15000` 이라는 조건을 사용하면 된다.

```
SELECT * FROM products WHERE price < 5000 OR price > 15000;
```

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

이럴 때 `NOT BETWEEN`을 쓰면 의도를 더 명확하게 표현할 수 있다. '이 범위에 속하지 않는 것'을 찾는다는 의미를 바로 알 수 있다.

[실습] products 테이블에서 가격이 5000원 이상 15000원 이하가 아닌 상품 조회하기

```
SELECT * FROM products WHERE price NOT BETWEEN 5000 AND 15000;
```

- 실행 결과는 이전과 같다.
- 이렇게 `NOT BETWEEN`을 사용하면 특정 범위를 제외한 데이터를 간결하게 조회할 수 있다.

IN: 목록에 포함된 값 찾기

이번에는 마케팅팀에서 요청이 왔다. **"갤럭시, 아이폰과 에어팟만 모아서 할인 행사를 하고 싶어요. 해당 상품들 목록만 뽑아주세요."**

```
SELECT * FROM products WHERE name = '갤럭시' OR name = '아이폰' OR name = '에어팟';
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

이 요청을 `OR`로 해결하려면 `WHERE name = '갤럭시' OR name = '아이폰' OR name = '에어팟'` 처럼 길게 써야 한다. 만약 상품이 50개라면 `OR`를 49번이나 써야 할까? 이럴 때 `IN`을 사용한다.

`IN (목록)` 구문은 괄호 안에 있는 목록 중 하나라도 일치하는 것이 있으면 선택한다.

[실습] products 테이블에서 이름이 '갤럭시', '아이폰', '에어팟' 중 하나인 상품 조회하기

```
SELECT * FROM products WHERE name IN ('갤럭시', '아이폰', '에어팟');
```

- 실행 결과는 이전과 같다.
- OR를 여러 번 쓰는 것보다 가독성이 좋다.

NOT IN: 목록에 포함되지 않은 값 찾기

`IN`의 반대도 당연히 필요하다. 왜일까?

마케팅팀에서 다시 요청했다. **"갤럭시, 아이폰, 에어팟은 행사 상품으로 지정했으니, 이 상품들을 제외한 나머지 상품들**

의 재고를 점검하고 싶습니다. 목록을 뽑아주세요."

이럴 때 `WHERE name != '갤럭시' AND name != '아이폰' AND name != '에어팟'` 처럼 `!=` 와 `AND` 를 반복해서 사용할 수도 있다.

```
SELECT * FROM products WHERE name != '갤럭시' AND name != '아이폰' AND name != '에어팟';
```

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
5	보급형 스마트폰	NULL	5000	100

이럴 때 `NOT IN` 을 사용하면 훨씬 간단해진다.

[실습] products 테이블에서 이름이 '갤럭시', '아이폰', '에어팟'이 아닌 상품 조회하기

```
SELECT * FROM products WHERE name NOT IN ('갤럭시', '아이폰', '에어팟');
```

- 실행 결과는 이전과 같다.
- 이렇게 특정 목록에 있는 것들을 제외하고 싶을 때 `NOT IN` 이 유용하게 사용된다.

LIKE: 문자열의 일부로 검색하기 (패턴 매칭)

CS팀이 다급한 요청을 한다.

"고객 한 분이 자기 이메일이 `sejong` 으로 시작하는 것까지만 기억난대요! `sejong` 으로 시작하는 이메일을 가진 고객을 찾아주세요."

`=` 연산자는 문자열 전체가 정확히 일치해야만 찾을 수 있다. 이처럼 문자열의 일부만으로 데이터를 검색하고 싶을 때 `LIKE` 연산자와 '와일드카드'를 함께 사용한다.

와일드카드 문자

- `%` (퍼센트): 0개 이상의 **모든 문자**를 의미한다.
  - `'sejong%'` : `sejong` 으로 시작하는 모든 문자열 (`sejong@example.com` , `sejong123` 등)

    - `'%@example.com'` : `@example.com`으로 끝나는 모든 문자열 (`aaa@example.com`, `hello@example.com` 등)
    - `'%서울%'` : `서울`이라는 단어를 포함하는 모든 문자열 (`수도서울`, `서울에 살자`, `수도 서울에 살자` 등)
- `_` (언더스코어): 정확히 **한 개의 문자**를 의미한다.
    - `'이_신'` : '이'로 시작하고 '신'으로 끝나는 세 글자 이름 (`이순신`, `이방신` 등, 예를 들어 `이나라신`은 정확히 한 개의 문자가 아니므로 탈락)

[실습] customers 테이블에서 이메일이 'sejong'으로 시작하는 고객 검색하기

```
SELECT * FROM customers WHERE email LIKE 'sejong%';
```

[실행 결과]

customer_id	name	email	password	address	join_date
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-00 00:00:00

NOT LIKE: 특정 패턴을 제외하고 검색하기

`LIKE`의 반대인 `NOT LIKE`는 어떤 상황에 사용할까?

[문제 상황] 마케팅 팀에서 요청이 왔다. **"서울특별시에 살지 않는 고객을 대상으로 마케팅을 하고 싶어요. 대상 고객 목록을 출력해주세요."**

[실습] 서울특별시에 살지 않는 고객 검색하기

```
SELECT * FROM customers WHERE address NOT LIKE '서울특별시%';
```

customer_id	name	email	password	address	join_date
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-01 00:00:00

- 서울특별시로 시작하는 고객을 제외한 나머지 고객을 확인할 수 있다.

여기서 설명하지 않은 `IS NULL`은 뒤에서 설명한다.

ORDER BY - 정렬

`WHERE`를 마스터해서 원하는 데이터를 필터링하는데 성공했다. 그런데 새로운 유형의 요청이 들어온다.

"최근에 가입한 고객 순서대로 명단을 뽑아주세요! 방금 가입한 사람이 맨 위에 보이게요."
"우리 쇼핑몰에서 가장 비싼 상품부터 순서대로 보여주세요! VIP 고객들에게 추천하게요."

`SELECT`와 `WHERE`만 사용해서 데이터를 조회하면, 그 결과는 어떤 순서로 나올까? 정답은 '알 수 없다' 또는 '데이터베이스 마음대로'이다. 데이터베이스는 데이터를 가장 효율적으로 저장하고 관리할 뿐, 우리가 조회할 때 특정 순서대로 보여줘야 할 의무가 없다. 어떨 때는 기본 키(`PRIMARY KEY`) 순서로, 어떨 때는 데이터가 삽입된 순서로 보이는 등 일관성이 없다.

따라서 데이터를 분석하거나 보고서를 만들 때는 의미 있는 순서로 나열하는 과정이 반드시 필요하다. '최신순', '가격 높은순', '이름 가나다순' 처럼 말이다. 이럴 때 사용하는 것이 `ORDER BY` 절이다.

결과의 순서를 지정하는 ORDER BY

```
SELECT 열이름
  FROM 테이블이름
 WHERE 조건
 ORDER BY 정렬기준열 [정렬방식];
```

- `[정렬방식]`은 생략 가능

`ORDER BY` 절은 `SELECT` 문의 가장 마지막에 위치하며, 조회된 결과를 특정 열의 값을 기준으로 정렬하는 역할을 한다.

정렬 방식에는 다음 두 가지가 있다.

- `ASC` (Ascending): **오름차순** 정렬. 작은 값에서 큰 값으로 (1, 2, 3... / A, B, C... / 오래된 날짜부터 최신 날짜

로).
    - `ORDER BY`의 기본값이므로 생략할 수 있다.
    - 점점 올라간다고 외우면 쉽다.
- `DESC` (Descending): **내림차순** 정렬. 큰 값에서 작은 값으로 (10, 9, 8... / Z, Y, X... / 최신 날짜부터 오래된 날짜로).
    - 내림차순을 원할 경우 반드시 명시해야 한다.
    - 점점 내려간다고 외우면 쉽다.

첫 번째 요청, "최근에 가입한 고객 순서대로" 명단을 만들어 보자. '최근'이라는 것은 가입일(`join_date`)이 가장 나중인 값이므로, `join_date` 열을 기준으로 **내림차순(`DESC`)** 정렬해야 한다.

[실습] customers 테이블을 가입일(join_date) 최신순으로 정렬하기

```
SELECT * FROM customers ORDER BY join_date DESC;
```

[실행 결과]

customer_id	name	email	password	address	join_date
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-00 00:00:00
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-00 00:00:00
1	이순신	yisunsin@example.com	password123	서울특별시 중구 세종대로	2023-05-00 00:00:00

가입일(`join_date`)이 최신인 '장영실'부터 가장 오래된 '이순신' 순서로 정렬된 것을 볼 수 있다.

이번에는 반대로 오래된 가입 고객 순서대로 오름차순(`ASC`)으로 출력해보자.

```
SELECT * FROM customers ORDER BY join_date ASC;
SELECT * FROM customers ORDER BY join_date; -- ASC는 생략 가능
```

- 정렬의 기본 값은 오름차순(`ASC`)이다. 따라서 오름차순일 경우 ASC를 생략할 수 있다.

[실행 결과]

customer_id	name	email	password	address	join_date
1	이순신	yisunsin@example.com	password123	서울특별시 중구 세종대로	2023-05-0 00:00:00
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-0 00:00:00
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-0 00:00:00

두 번째 요청, "가격이 낮은 상품부터" 보여달라고 한다면 가격(`price`)을 기준으로 **오름차순(`ASC`)** 정렬하면 된다. `ASC`는 생략 가능하므로 `ORDER BY price` 라고만 써도 된다.

[실습] products 테이블을 가격(price) 낮은순으로 정렬하기

```
SELECT * FROM products ORDER BY price ASC; -- 'ASC'는 생략 가능
```

[실행 결과]

product_id	name	description	price	stock_quantity
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
5	보급형 스마트폰	NULL	5000	100
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

가격이 3000원인 '에어팟'부터 20000원인 'LG 그램' 순서로 완벽하게 정렬되었다.

> ✨ **실무 팁**
> 오름차순인 경우 ASC 키워드는 생략할 수 있다. 정렬의 기본 값이기 때문이다.

> 실무에서는 오름차순인 경우에는 ASC 키워드를 대부분 생략하고 ORDER BY 만 사용한다.
> 물론 내림차순의 경우 DESC를 필수로 적어야 하기 때문에 생략하면 안된다.

정렬 기준이 여러 개라면?

`ORDER BY`로 데이터를 정렬하니 보고서가 한결 깔끔해졌다. 그러자 좀 더 까다로운 요청이 들어온다.

"상품 목록을 일단 재고 수량(`stock_quantity`)이 많은 순서대로 정렬해 주세요. 그런데 만약 재고 수량이 같다면, 그 상품들끼리는 가격(`price`)이 낮은 순서대로 정렬해주세요."

1차 정렬 기준은 '재고 수량'이고, 1차 정렬 기준의 값이 같을 경우에만 적용되는 2차 정렬 기준 '가격'이 필요하다. 이처럼 정렬 기준이 여러 개일 때는 어떻게 해야 할까?

다중 열 정렬 (Multi-column Sort)

아주 간단하다. `ORDER BY` 절에 여러 개의 열 이름을 콤마(`,`)로 구분하여 나열하면 된다. 정렬의 우선순위는 당연히 먼저 쓴 열부터 적용된다.

```
ORDER BY 1차정렬열 [정렬방식], 2차정렬열 [정렬방식], ...
```

각 열마다 `ASC`나 `DESC`를 개별적으로 지정할 수도 있다.

요청을 해결해 보자. 1차 기준은 재고 수량(`stock_quantity`) 내림차순(`DESC`), 2차 기준은 가격(`price`) 오름차순(`ASC`)이다.

[실습] products 테이블을 재고수량 내림차순, 가격 오름차순으로 정렬하기

```
SELECT * FROM products ORDER BY stock_quantity DESC, price ASC;
-- price의 ASC는 생략 가능
```

해당 SQL 쿼리는 상품 데이터를 **재고 수량(`stock_quantity`)이 많은 순서**로 먼저 정렬하고, 만약 재고 수량이 같다면 **가격(`price`)이 낮은 순서**로 다시 정렬한다.

[실행 결과]

product_id	name	description	price	stock_quantity
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
5	보급형 스마트폰	NULL	5000	100
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

결과를 자세히 보자. '갤럭시'와 '아이폰'은 재고 수량이 55개로 동일하다. 이 경우 2차 정렬 기준인 `price ASC`가 적용되어, 가격이 더 낮은 '아이폰'(5000원)이 '갤럭시'(10000원)보다 먼저 표시된다.

LIMIT - 개수 제한

이제 우리는 원하는 데이터를, 원하는 순서대로 가져올 수 있게 되었다. 그때, 또다시 대표가 우리를 찾는다.
"우리 쇼핑몰에서 가장 비싼 상품 TOP 2만 딱 알려주게! 전부 다는 필요 없어."

또 다른 팀에서는 이런 요청이 온다.
"상품 목록을 웹 페이지에 보여줘야 하는데, 한 페이지에 다 보여주기엔 너무 많아요! 20개씩 끊어서 보여주고 싶습니다. 1페이지, 2페이지 넘어가는 기능이 필요해요."

데이터가 수백만, 수천만 건이 된다고 상상해 보자. 이 데이터를 `SELECT` 문으로 한 번에 모두 조회하는 것은 시스템에 엄청난 부담을 준다. 조회 속도가 느려지는 것은 물론이고, 심하면 서버가 다운될 수도 있다. 네트워크에도 부하가 걸린다.
사실 우리는 전체 데이터가 아니라 '상위 N개'만 보거나, '특정 구간'의 데이터만 잘라서 보고 싶은 경우가 훨씬 많다. 이럴 때 사용하는 것이 바로 `LIMIT` 절이다.

조회 결과 개수를 제한하는 LIMIT

`LIMIT`는 `ORDER BY` 뒤에 위치하며, 조회할 결과의 최대 개수를 제한하는 역할을 한다.

```
SELECT 열이름
FROM 테이블이름
WHERE 조건
ORDER BY 정렬기준
LIMIT 개수;
```

여기서 매우 중요한 점이 있다. `LIMIT`는 `ORDER BY`와 함께 사용해야 진정한 의미가 있다. 왜일까? 앞에서 말했듯이 데이터베이스는 결과의 순서를 보장하지 않는다. 정렬되지 않은 상태에서 `LIMIT 2`을 쓴다면, 실행할 때마다 다른 2개의 데이터가 나올 수도 있다. '가장 비싼 상품 TOP 2'처럼 의미 있는 상위 N개를 뽑으려면, **반드시 먼저 `ORDER BY`로 정렬한 뒤 `LIMIT`로 잘라내야 한다.**

대표의 요청을 해결해 보자. '가장 비싼 상품 2개'를 뽑으려면, 먼저 가격(`price`)을 내림차순(`DESC`)으로 정렬한 뒤, 상위 2개만 가져오면 된다.

[실습] products 테이블에서 가장 비싼 상품 2개만 조회하기

```
SELECT * FROM products ORDER BY price DESC LIMIT 2;
```

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55

가장 비싼 'LG 그램'부터 순서대로 2개의 상품만 정확히 조회되었다.

실무의 핵심, 페이징(Pagination) 처리 [문제 상황]

게시판, 상품 목록, 고객 리스트 등 웹사이트나 앱에서 보는 거의 모든 목록 페이지는 '페이징(Paging)' 처리가 되어 있다. 수십만 개의 게시글을 한 페이지에 다 보여주는 사이트는 없다. 보통 10개나 20개씩 끊어서 보여주고, 하단에 [1], [2], [3], [다음], [마지막] 같은 페이지 번호를 제공한다.

사용자가 2페이지를 클릭하면 1페이지에 보였던 데이터는 건너뛰고, 그 다음 데이터들을 보여줘야 한다. 이 페이징 기능은 어떻게 구현할까? 바로 `LIMIT`의 또 다른 사용법, '오프셋(Offset)'을 이용하면 된다.

특정 범위의 결과만 조회하기: LIMIT, 오프셋(Offset)

`LIMIT`는 숫자 하나만 쓸 수도 있지만, 두 개를 쓸 수도 있다.

```
LIMIT 건너뛸개수(offset), 가져올개수(row_count);
```

- `offset`: 앞에서부터 몇 개의 데이터를 건너뛸 것인지를 지정한다.
- `row_count`: 건너뛴 다음부터 몇 개의 데이터를 가져올 것인지를 지정한다.

예를 들어, `LIMIT 10, 5` 라고 하면, 맨 앞의 10개 데이터는 건너뛰고, 11번째 데이터부터 5개를 가져오라는 의미다.

> **LIMIT 다양한 문법**
>
> - MySQL에서는 `LIMIT 0, 2`와 같이 `offset`이 0인 경우 `LIMIT 2`와 같이 줄여 쓸 수 있다.
> - MySQL에서는 `LIMIT 0, 2`를 `LIMIT 2 OFFSET 0`과 같이 `OFFSET` 키워드를 넣어 사용할 수 있다.

이것을 페이징에 적용해 보자. 한 페이지에 2개의 상품을 보여준다고 가정한다.
(보통은 10개 정도는 보여주는데, 샘플 데이터가 적으므로 2개로 하겠다.)

- **1페이지**: 처음부터 2개를 가져와야 한다. 건너뛸 필요가 없으니 `offset`은 `0`. `LIMIT 0, 2`
- **2페이지**: 앞의 2개(1페이지 분량)를 건너뛰고 2개를 가져와야 한다. `offset`은 `2`. `LIMIT 2, 2`
- **3페이지**: 앞의 4개(1, 2페이지 분량)를 건너뛰고 2개를 가져와야 한다. `offset`은 `4`. `LIMIT 4, 2`

여기서 규칙이 보이는가? `offset`을 계산하는 간단한 공식이 있다.

> `offset = (페이지번호 - 1) * 페이지당_게시물수`

이 공식 하나만 알면 필요한 페이징을 구현할 수 있다.

[실습] products 목록을 한 페이지당 2개씩 보여줄 때, 1페이지 조회하기

- `offset = (1 - 1) * 2 = 0`. `LIMIT 0, 2`가 된다.

사용자에게 일관된 순서를 보여주기 위해 `product_id`로 정렬하는 것을 잊지 말자.

```
SELECT * FROM products ORDER BY product_id ASC LIMIT 0, 2;
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

[실습] products 목록의 2페이지 (3~4번째 상품) 조회하기

`offset` = `(2 - 1) * 2 = 2`. `LIMIT 2, 2`가 된다.

```
SELECT * FROM products ORDER BY product_id ASC LIMIT 2, 2;
```

[실행 결과]

product_id	name	description	price	stock_quantity
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

[실습] products 목록의 3페이지 (5번째 상품) 조회하기

`offset` = `(3 - 1) * 2 = 4`. `LIMIT 4, 2`가 된다.

```
SELECT * FROM products ORDER BY product_id ASC LIMIT 4, 2;
```

[실행 결과]

product_id	name	description	price	stock_quantity

5	보급형 스마트폰	NULL	5000	100

- 마지막 페이지에서는 가져올 데이터가 1개뿐이므로 `LIMIT`에 지정한 `2`개보다 적은 1개의 행만 반환되었다.

이처럼 `LIMIT`와 `offset`을 이용하면 아무리 많은 데이터라도 잘게 쪼개서 효율적으로 사용자에게 보여줄 수 있다. 페이징은 실무에서 반드시 알아야 하는 핵심 기능이다.

DISTINCT - 중복 제거

데이터를 조회하다 보면, 우리가 원하지 않는 중복된 값들이 계속 나타나는 경우가 있다. 이럴 때 어떻게 유일한 값들만 깔끔하게 추려낼 수 있을까?

우리 쇼핑몰이 점점 잘 되고 있어서 주문 데이터가 차곡차곡 쌓이고 있다. 어느 날, 마케팅 팀에서 이런 요청을 해왔다. "우리 쇼핑몰에서 **한 번이라도 주문을 한 고객의 ID 목록**을 뽑아주세요. 고객들에게 감사 이벤트를 하고싶어요."

우리는 주문 정보를 담고 있는 `orders` 테이블을 살펴보기로 했다. `orders` 테이블에는 어떤 고객이 주문했는지 알려주는 `customer_id` 컬럼이 있다. 간단한 `SELECT` 문으로 해결할 수 있을 것 같다.

```
SELECT customer_id FROM orders;
```

이 쿼리를 실행하면 어떤 결과가 나올까? `orders` 테이블에 있는 모든 `customer_id`를 그대로 가져올 것이다.

[실행 결과]

customer_id
1
1
2
2

3

결과를 보니 뭔가 이상하지 않은가? 마케팅 팀에서 원한 것은 '주문한 고객 목록'인데, 이 결과에는 `1`번 고객과 `2`번 고객이 각각 두 번씩이나 나타난다. 이것은 '주문이 들어온 목록'이지 '주문한 고객의 유일한 목록'이 아니다. 이순신 고객(ID: 1)이 두 번 주문했고, 세종대왕 고객(ID: 2)도 두 번 주문했기 때문에 이런 결과가 나온 것이다. 이 데이터를 그대로 마케팅 팀에 전달하면 "어? 고객이 총 5명인가요?" 하고 오해할 수 있다.

바로 이럴 때, 우리는 중복된 값을 제거하고 순수하게 '어떤 종류'의 데이터가 있는지 확인할 수 있어야 한다.

해결책 - DISTINCT

이런 문제 상황을 해결하기 위해 SQL은 아주 간단한 해결책을 제공한다. 바로 `DISTINCT` 키워드다.

`DISTINCT`는 `SELECT` 문에서 컬럼 이름 바로 앞에 붙여 사용하며, 조회된 결과에서 중복된 행을 모두 제거하고 유일한(unique) 값만 남겨준다.

사용법은 아주 간단하다.

```
SELECT DISTINCT 컬럼명 FROM 테이블명;
```

문제 상황을 `DISTINCT`를 사용해서 다시 해결해 보자. 주문한 고객의 ID를 중복 없이 보고 싶으니, `customer_id` 컬럼에 `DISTINCT`를 적용하면 된다.

```
SELECT DISTINCT customer_id FROM orders;
```

[실행 결과]

customer_id
1
2
3

결과를 보자. 이제 각 고객 ID가 한 번씩만 나타난다. `1`, `2`, `3`번 고객이 우리 쇼핑몰에서 주문을 했다는 사실을 명확하게 알 수 있다. 이것이 바로 마케팅 팀이 원했던 '유일한 고객 ID 목록'이다. `DISTINCT` 키워드 하나로 문제가 아주 간단하게 해결되었다.

여러 컬럼에 DISTINCT 적용하기

`DISTINCT`는 하나의 컬럼에만 사용할 수 있는 것이 아니다. 여러 개의 컬럼을 대상으로도 사용할 수 있다. 이 경우, 지정된 **모든 컬럼의 조합**이 유일한 것들만 보여준다.

예를 들어, "어떤 고객이 어떤 상품을 구매했는지 그 조합을 중복 없이 보고 싶다"는 새로운 요청이 왔다고 가정해 보자.

먼저 `DISTINCT` 없이 `orders` 테이블을 모두 조회해보자.

```
SELECT customer_id, product_id FROM orders;
```

[실행 결과]

customer_id	product_id
1	1
2	2
3	3
1	4
2	2

여기서 보면 `(2, 2)` 라는 조합, 즉 세종대왕 고객이 LG 그램을 주문한 기록이 두 번 나타난다. 만약 우리가 '고객-상품'의 유일한 구매 조합만 보고 싶다면 이 중복은 제거하는 것이 좋다.

이제 `DISTINCT`를 사용해 보자.

```
SELECT DISTINCT customer_id, product_id FROM orders;
```

`DISTINCT`는 `customer_id`와 `product_id` 두 컬럼을 하나의 묶음으로 보고, 이 묶음이 중복되는지를 판단한다.

[실행 결과]

customer_id	product_id
1	1
2	2
3	3
1	4

결과를 보면, 중복이었던 `(2, 2)` 행이 하나로 합쳐져 총 4개의 유일한 '고객-상품' 조합만 남은 것을 확인할 수 있다. 이처럼 `DISTINCT`는 여러 컬럼에 걸쳐 중복을 제거할 때도 매우 유용하다.

> ✨ **실무 팁**
> 실무에서 `DISTINCT`는 데이터를 탐색하고 분석할 때 많이 사용된다. "우리 서비스에는 어떤 종류의 고객들이 있지?", "판매된 상품의 카테고리는 총 몇 가지지?" 와 같이 데이터의 '종류'를 파악할 때 아주 유용하다.
>
> 다만 한 가지 기억해야 할 점이 있다. `DISTINCT`는 결과를 보여주기 전에 내부적으로 중복을 제거하기 위한 과정을 거쳐야 한다. 따라서 데이터가 아주 많은 경우 일반적인 `SELECT`보다 성능이 저하될 수 있다. 대량의 데이터라면 성능을 확인할 필요가 있다. 물론, 대부분의 경우에는 크게 문제 되지 않으니 걱정 말고 사용해도 괜찮다.

NULL - 알 수 없는 값

이번 시간에는 SQL을 처음 다룰 때 많이 실수할 수 있는 `NULL`에 대해 자세히 알아보자.

NULL 비교

상품 관리 팀장이 우리에게 와서 묻는다. **"쇼핑몰에 상품을 등록하다 보니 몇몇 상품의 설명을 빠뜨린 것 같아요. 상품 설명이 아직 등록되지 않은 상품들만 골라서 보여주세요."**

현재 `보급형 스마트폰`의 `description` 값이 `NULL`이다. 따라서 이 상품을 조회하면 된다.

`description`이 `NULL`인 상품을 `=` 비교를 통해 조회해보자.

```
SELECT * FROM products WHERE description = NULL
```

[실행 결과]

- 없음

분명 `보급형 스마트폰`의 `description` 값은 `NULL`이기 때문에 조회가 될 것으로 기대했는데, 기대와는 다르게 아무 상품도 조회되지 않았다.

NULL 연산 - 알 수 없음

데이터베이스에서 '값이 없음'을 나타내는 특별한 상태를 `NULL`이라고 한다. `NULL`은 숫자 `0`이나 빈 문자열(`''`)과는 완전히 다른 개념이다. 말 그대로 '알 수 없는 값', '존재하지 않는 값'이다.
여기서 아주 중요한 함정이 있다. 상품 설명이 비어있는 상품을 찾기 위해 `WHERE description = NULL` 이라고 쓰면 될까? **절대 안된다.** `NULL`은 특정 값이 아니기 때문에 등호(`=`)로 비교할 수 없다.

이는 SQL에서 `NULL`이 '값이 없는 상태'를 의미하는 특별한 존재이기 때문이다. `NULL`은 0도 아니고, 공백 문자열('')도 아니다. '알 수 없는 값'이라는 개념에 가깝다.

좀 더 쉽게 풀어서 이야기하자면
NULL은 '빈 상자'가 아니라 '상자가 있는지조차 모른다'는 표시다.

- `price=0` : 숫자가 `0`이라는 값이 있다.
- `description=''` : 글자가 없다는 값이 있다. (두 쉼표가 사용되었다.)
- `NULL` : 값이 있는지 자체를 모른다.

따라서 `어떤 값 = NULL` 이라는 비교 연산은 항상 '알 수 없음(UNKNOWN)'이라는 결과를 반환한다. `NULL = NULL` 조차도 참(TRUE)이 아닌 '알 수 없음(UNKNOWN)'이다. **비교는 양쪽이 다 값을 가질 때만 참 거짓을 결정할 수 있다.**
`WHERE` 절은 조건의 결과가 '참(TRUE)'인 행만 반환하므로, '알 수 없음(UNKNOWN)'으로 판별된 행은 결과에 포함시키지 않는다.

이런 문제를 해결하기 위해 SQL은 `IS NULL`이라는 특별한 키워드를 제공한다.
`NULL` 인지 아닌지를 검사하기 위해서는 반드시 `IS NULL` 또는 `IS NOT NULL`을 사용해야 한다.

- `IS NULL` : 해당 열의 값이 `NULL`인 행을 찾는다.
- `IS NOT NULL` : 해당 열의 값이 `NULL`이 아닌, 즉 데이터가 입력된 행을 찾는다.

[실습] products 테이블에서 상품 설명(description)이 없는(NULL) 상품 조회하기

우리가 추가한 '보급형 스마트폰'의 `description`은 `NULL`이다. 이 데이터를 `IS NULL`을 이용해 정확히 찾아낼 수 있다.

```
SELECT * FROM products WHERE description IS NULL;
```

[실행 결과]

product_id	name	description	price	stock_quantity
5	보급형 스마트폰	NULL	5000	100

반대로 설명이 있는 상품만 찾으려면 `IS NOT NULL`을 사용한다.

```
SELECT * FROM products WHERE description IS NOT NULL;
```

[실행 결과]

product_id	name	description	price	stock_quantity
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

이처럼 `IS NULL` 연산자를 사용해야만 `NULL` 상태인 데이터를 정확히 찾아낼 수 있다. 이는 SQL을 처음 배울 때 가장 흔하게 하는 실수 중 하나이므로 **"NULL을 비교할 때는 `=`이 아니라 `IS`를 사용한다"**는 점을 반드시 기억하자.

NULL 정렬

왜 NULL 값의 정렬을 알아야 할까?

우리가 다루는 데이터가 항상 완벽하게 채워져 있지는 않다. 예를 들어, `products` 테이블을 다시 보자. `description` 열은 `NULL`을 허용한다. 실제로 '보급형 스마트폰'의 `description` 값은 `NULL`이다.

이때, **"상품 설명을 기준으로 가나다순 정렬해주세요."** 라는 요청을 받으면 어떻게 될까? `NULL`은 문자도, 숫자도 아닌데 도대체 어디에 위치해야 할까? `ORDER BY`는 이 `NULL`을 어떻게 처리할까? 이 규칙을 모르면 데이터가 왜 특정 위치에 나오는지 설명할 수 없고, 원하는 결과를 얻기 위해 쿼리를 수정할 수도 없다.

MySQL의 NULL 정렬 규칙

결론부터 말하자면, **MySQL은 `NULL`을 가장 작은 값으로 취급한다.** 이것만 기억하면 된다.

- `ORDER BY column ASC` (오름차순): `NULL` 값이 가장 **먼저** 나온다. (가장 작은 값으로 취급되므로)
- `ORDER BY column DESC` (내림차순): `NULL` 값이 가장 **나중에** 나온다. (가장 작은 값이라 맨 아래로 밀려나므로)

이것은 데이터베이스 시스템마다 정책이 다를 수 있으므로(Oracle은 `NULL`을 가장 큰 값으로 취급한다), 내가 사용하는 DB가 어떤 규칙을 따르는지 명확히 아는 것이 중요하다.

예제를 통해 확인하기

자, `products` 테이블의 `description` 열을 가지고 직접 확인해 보자.

[실습] `description` 열을 오름차순으로 정렬하기

```
SELECT * FROM products ORDER BY description ASC;
```

[실행 결과]

product_id	name	description	price	stock_quantity
5	보급형 스마트폰	NULL	5000	100
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

- `NULL` 값은 가장 작은 값이다. 예상대로 `description`이 `NULL`인 '보급형 스마트폰'이 가장 먼저 나왔다.
- `description ASC`로 정렬했으므로 ㅈ ➔ ㅊ ➔ ㅍ 순서로 정렬된다.

[실습] `description` 열을 내림차순으로 정렬하기

이번에는 내림차순으로 정렬해 보자.

```
SELECT * FROM products ORDER BY description DESC;
```

[실행 결과]

product_id	name	description	price	stock_quantity
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
5	보급형 스마트폰	NULL	5000	100

- 내림차순으로 정렬하니, 가장 작은 `NULL` 값이 가장 마지막에 위치하는 것을 확인할 수 있다.
- `description DESC`로 정렬했으므로 ㅈ ➔ ㅊ ➔ ㅍ 순서의 반대인 ㅍ ➔ ㅊ ➔ ㅈ 순서로 정렬된다.

[실무 팁] NULL 위치를 강제로 바꾸고 싶을 때

실무에서는 "상품 설명을 내림차순으로 정렬하되, 설명이 없는 상품(`NULL`)은 빨리 확인할 수 있게 맨 앞으로 보내주세요." 와 같이 기본 동작과 다른 요구사항이 종종 있다. 이럴 때는 어떻게 할까?
앞서 확인했듯이 NULL은 가장 작은 값이기 때문에 내림차순으로 정렬하면 마지막에 출력된다.

이때는 `ORDER BY` 절에 조건을 추가하는 트릭을 사용한다. `IS NULL`을 활용하는 것이다.

먼저 다음 쿼리를 실행해보자.

```
SELECT product_id, name, description, description IS NULL
FROM products
ORDER BY description DESC;
```

- 앞서 사용한 쿼리의 SELECT에 `description IS NULL`을 추가했다.

[실행 결과]

product_id	name	description	description IS NULL
4	에어팟	편리한 사용성의 무선 이어폰	0
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	0
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	0
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	0
5	보급형 스마트폰	NULL	1

- `description IS NULL` 필드를 확인해보자. `description IS NULL` 조건에 만족하는 경우에는 1을 반환한다. 나머지는 0을 반환한다.
- `description DESC`로 정렬했으므로 ㅈ ➔ ㅊ ➔ ㅍ 순서의 반대인 ㅍ ➔ ㅊ ➔ ㅈ 순서로 정렬된다.

> **MySQL은 TRUE는 1, FALSE는 0이라는 숫자로 표현한다.**

`(열_이름 IS NULL)` 이라는 조건은 해당 열의 값이 `NULL`이면 `1`(`TRUE`)을, `NULL`이 아니면 `0`(`FALSE`)을 반환한다.
이를 이용해서 정렬하면 된다. 1값은 0보다 크다!

```
SELECT product_id, name, description, description IS NULL
FROM products
ORDER BY description IS NULL DESC;
```

- `description IS NULL`을 정렬 조건에 사용했다.

[실행 결과]

product_id	name	description	description IS NULL
5	보급형 스마트폰	NULL	1
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	0
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	0
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	0
4	에어팟	편리한 사용성의 무선 이어폰	0

- `description IS NULL`을 만족하는 **보급형 스마트폰**의 값이 1이고, 나머지는 그보다 작은 0이다.
- 이 순서로 내림차순 정렬하면 된다.

원하는 `NULL` 값이 가장 먼저 출력되었다. 그런데 한 가지 문제가 남아있다.
요구사항을 다시 확인해보자.

"**상품 설명을 내림차순으로 정렬**하되, 설명이 없는 상품(`NULL`)은 빨리 확인할 수 있게 맨 앞으로 보내주세요."

상품 설명을 내림차순으로 정렬해야 한다. 따라서 ㅈ ➔ ㅊ ➔ ㅍ 순서의 반대인 ㅍ ➔ ㅊ ➔ ㅈ 순서로 정렬되어야 한다.
그런데 상품은 `description` 순서가 아니라 `description IS NULL` 순서로 정렬되어 있는 것이다. 둘의 순서는 다르다.

이럴 때 앞서 배운 다중 열 정렬을 사용하면 된다.

```
SELECT product_id, name, description, description IS NULL
FROM products
ORDER BY description IS NULL DESC, description DESC;
```

- `description IS NULL DESC`을 첫 번째 정렬 조건으로
- `description DESC`을 두 번째 정렬 조건으로 사용한다.

[실행 결과]

product_id	name	description	description IS NULL
5	보급형 스마트폰	NULL	1
4	에어팟	편리한 사용성의 무선 이어폰	0
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	0
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	0
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	0

- 1순위인 `description IS NULL`에서 보급형 스마트폰의 값만 1이다. 따라서 가장 상단에 출력된다.
- 나머지 값들은 `description IS NULL`의 값이 모두 0이다. 1순위 값이 0으로 같으므로 2순위인 `description DESC`를 통해 순서를 조정한다.
- 여기서는 내림차순으로 정렬했으므로 ㅈ ➜ ㅊ ➜ ㅍ 순서의 반대인 ㅍ ➜ ㅊ ➜ ㅈ 순서로 정렬되었다.

`ORDER BY (description IS NULL) DESC` 라는 1차 정렬 기준 덕분에 `NULL` 값이 (결과 `1`)이 `NULL`이 아닌 값들(결과 `0`)보다 먼저 오게 되었다. 그리고 `description DESC`라는 2차 정렬 기준으로 `NULL`이 아닌 값들끼리 정렬된 것을 볼 수 있다. 이런 식으로 `NULL`의 위치를 자유자재로 제어할 수 있다.

문제와 풀이

문제1: 특정 열 조회 및 별칭 사용

[문제]

`products` 테이블에 있는 모든 상품의 이름(`name`)과 가격(`price`) 정보를 조회해라. 단, 조회 결과의 열 이름은 각각 '상품명'과 '판매가'로 표시되어야 한다.

[실행 결과]

상품명	판매가
갤럭시	10000
LG 그램	20000

아이폰	5000
에어팟	3000
보급형 스마트폰	5000

[정답]

```
SELECT name AS 상품명, price AS 판매가 FROM products;
```

문제2: 간단한 조건으로 필터링하기

[문제]

`customers` 테이블에서 '장영실' 고객의 모든 정보를 조회해라.

[실행 결과]

customer_id	name	email	password	address	join_date
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-01 00:00:00

[정답]

```
SELECT * FROM customers WHERE name = '장영실';
```

문제3: 복합 조건으로 필터링하기 (AND, OR)

[문제]

`products` 테이블에서 가격(`price`)이 10000원 이상이면서, 동시에 재고(`stock_quantity`)가 50개 미만인 상품을 조회해라.

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

[정답]

```sql
SELECT * FROM products WHERE price >= 10000 AND stock_quantity < 50;
```

문제4: 특정 범위 및 목록으로 필터링하기 (BETWEEN, IN)

[문제]

`products` 테이블에서 `product_id`가 2번, 3번, 4번 중 하나에 해당하는 상품들의 이름(`name`)과 가격(`price`)을 조회해라.

[실행 결과]

name	price
LG 그램	20000
아이폰	5000
에어팟	3000

[정답]

`IN` 연산자를 사용하여 특정 목록에 포함되는 데이터를 조회한다.

```sql
SELECT name, price FROM products WHERE product_id IN (2, 3, 4);
```

문제5: 문자열 패턴으로 검색하기 (LIKE)

[문제]

`customers` 테이블에서 주소(`address`)가 '서울특별시'로 시작하는 고객의 이름(`name`)과 전체 주소(`address`)를 조회해라.

[실행 결과]

name	address
이순신	서울특별시 중구 세종대로
세종대왕	서울특별시 종로구 사직로

[정답]

`LIKE`와 와일드카드 `%`를 사용하여 '서울특별시'로 시작하는 모든 주소를 찾는다.

```sql
SELECT name, address FROM customers WHERE address LIKE '서울특별시%';
```

문제6: NULL 값 데이터 조회하기 (IS NULL)

[문제]

`products` 테이블에서 상품 설명(`description`)이 아직 입력되지 않은(NULL) 상품의 모든 정보를 조회해라.

[실행 결과]

product_id	name	description	price	stock_quantity

5	보급형 스마트폰	NULL	5000	100

[정답]

`NULL` 값을 비교할 때는 등호(`=`)가 아닌 `IS NULL` 연산자를 사용해야 한다.

```
SELECT * FROM products WHERE description IS NULL;
```

문제7: 결과 정렬하기 (ORDER BY)

[문제]

`products` 테이블의 모든 상품 정보를 가격(`price`)이 비싼 순서(내림차순)대로 정렬하여 조회해라.

[실행 결과]

product_id	name	description	price	stock_quantity
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
5	보급형 스마트폰	NULL	5000	100
4	에어팟	편리한 사용성의 무선 이어폰	3000	110

[정답]

`ORDER BY` 절에 정렬 기준 열을 명시하고, 내림차순을 위해 `DESC` 키워드를 사용한다.

```sql
SELECT * FROM products ORDER BY price DESC;
```

문제8: 다중 기준으로 정렬하기

[문제]

`products` 테이블의 상품 정보를 먼저 가격(`price`)의 오름차순으로 정렬하고, 만약 가격이 같다면 재고 수량(`stock_quantity`)이 많은 순(내림차순)으로 정렬하여 조회해라.

[실행 결과]

product_id	name	description	price	stock_quantity
4	에어팟	편리한 사용성의 무선 이어폰	3000	110
5	보급형 스마트폰	NULL	5000	100
3	아이폰	직관적인 사용자 경험을 제공하는 스마트폰	5000	55
1	갤럭시	최신 AI 기능이 탑재된 고성능 스마트폰	10000	55
2	LG 그램	초경량 디자인과 강력한 성능을 자랑하는 노트북	20000	35

[정답]

`ORDER BY` 절에 1차 정렬 기준(`price ASC`)과 2차 정렬 기준(`stock_quantity DESC`)을 콤마로 구분하여 순서대로 나열한다.

```sql
SELECT * FROM products ORDER BY price ASC, stock_quantity DESC;
```

문제9: 조회 결과 개수 제한하기 (LIMIT)

[문제]

`customers` 테이블에서 가장 최근에 가입한 고객 2명의 모든 정보를 조회해라.

[실행 결과]

customer_id	name	email	password	address	join_date
3	장영실	youngsil@example.com	password789	부산광역시 동래구 복천동	2025-05-00:00:00
2	세종대왕	sejong@example.com	password456	서울특별시 종로구 사직로	2024-05-00:00:00

[정답]

가입일(`join_date`)을 내림차순(`DESC`)으로 정렬하여 최신 가입자 순으로 만든 뒤, `LIMIT`를 이용해 상위 2개의 결과만 가져온다.

```
SELECT * FROM customers ORDER BY join_date DESC LIMIT 2;
```

문제10: 중복 값 제거하여 조회하기 (DISTINCT)

[문제]

`orders` 테이블을 참조하여, 한 번이라도 주문을 한 적이 있는 고객의 ID(`customer_id`)와 주문한 상품의 ID(`product_id`) 조합을 중복 없이 조회해라.

[실행 결과]

customer_id	product_id
1	1
2	2
3	3

1	4

[정답]

`SELECT` 절에 `DISTINCT` 키워드를 사용하여 `customer_id`와 `product_id`의 유일한 조합을 조회한다.

```
SELECT DISTINCT customer_id, product_id FROM orders;
```

문제11: 종합 실전 문제

[문제]

`products` 테이블에서 가격이 3000원을 초과하고 재고가 100개 이하인 상품들을 대상으로, 재고가 많은 순서대로 정렬하여 상위 3개의 상품명과 재고 수량을 조회해라.
이때 상품명은 '상품 이름'으로, 재고 수량은 '남은 수량'으로 출력해라.

[실행 결과]

상품 이름	남은 수량
보급형 스마트폰	100
갤럭시	55
아이폰	55

> ✨ **별칭에 공백( ) 같은 특수문자를 사용할 때는 백틱(`)으로 감싸면 된다.**

[정답]

`WHERE`로 조건을 필터링하고, `ORDER BY`로 정렬한 뒤, `LIMIT`로 개수를 제한한다. `SELECT`에서는 `AS`를 사용해 별칭을 지정한다.

```
```

```sql
SELECT
    name AS `상품 이름`,
    stock_quantity AS `남은 수량`
FROM
    products
WHERE
    price > 3000 AND stock_quantity <= 100
ORDER BY
    stock_quantity DESC
LIMIT 3;
```

정리

SELECT - 조회

- 데이터베이스에서 데이터를 꺼내보는 행위를 조회라 하며 `SELECT` 명령어를 사용한다.
- `SELECT * FROM 테이블명` 은 해당 테이블의 모든 열 데이터를 조회한다.
- `SELECT 열1, 열2 FROM 테이블명` 은 지정된 열의 데이터만 조회하여 성능과 가독성을 높인다.
- `AS` 키워드를 사용해 `SELECT name AS 고객명` 처럼 조회 결과의 열 이름에 별칭을 붙일 수 있다.

WHERE - 기본 검색

- `WHERE` 절은 특정 조건을 만족하는 데이터 행만 필터링하는 역할을 한다.
- `=`, `!=`, `>`, `<` 와 같은 비교 연산자를 사용하여 조건을 명시한다.
- `AND` `OR` `NOT` 과 같은 논리 연산자를 사용해 여러 조건을 조합할 수 있다.
- `AND` 는 모든 조건이 참일 때 `OR` 는 조건 중 하나라도 참일 때 결과를 반환한다.

WHERE - 편리한 조건 검색

- `BETWEEN a AND b` 는 a와 b 사이의 값(포함)을 검색하여 범위를 지정할 때 유용하다.
- `IN (목록)` 은 목록에 포함된 값 중 하나와 일치하는 데이터를 검색한다.
- `LIKE` 연산자는 `%` (0개 이상의 문자) `_` (정확히 한 개의 문자) 와일드카드와 함께 문자열의 일부로 검색할 때 사용한다.
- `IS NULL` 은 값이 `NULL` 인 데이터를 찾을 때 사용하며 `=` 연산자로는 `NULL` 을 비교할 수 없다.

ORDER BY - 정렬

- `ORDER BY` 절은 조회된 결과를 특정 열의 값을 기준으로 정렬한다.

- `ASC`는 오름차순(기본값 생략 가능) `DESC`는 내림차순을 의미한다.
- 콤마로 여러 열을 나열하여 다중 정렬을 할 수 있으며 명시된 순서대로 정렬 우선순위가 적용된다.
- 예시로 `ORDER BY 재고 DESC, 가격 ASC`는 재고를 내림차순으로 먼저 정렬하고 재고가 같으면 가격을 오름차순으로 정렬한다.

LIMIT - 개수 제한

- `LIMIT` 절은 조회 결과의 개수를 제한하며 보통 `ORDER BY`와 함께 사용해야 의미 있는 결과를 얻는다.
- `LIMIT 2`는 정렬된 결과에서 상위 2개의 데이터만 조회한다.
- `LIMIT 시작점, 개수`는 웹 페이지의 페이징 기능 구현에 핵심적으로 사용된다.
- `시작점 = (페이지번호 - 1) * 페이지당_개수` 공식을 통해 특정 페이지의 데이터를 조회할 수 있다.

DISTINCT - 중복 제거

- `DISTINCT` 키워드는 `SELECT` 조회 결과에서 중복된 행을 제거하고 고유한 값만 남긴다.
- `SELECT DISTINCT customer_id FROM orders`는 주문한 고객의 ID를 중복 없이 보여준다.
- 여러 컬럼에 적용하면 지정된 모든 컬럼의 조합이 유일한 데이터만 조회한다.

NULL - 알 수 없는 값

- `NULL`은 '알 수 없는 값' 또는 '존재하지 않는 값'을 의미하며 `0`이나 공백 문자와 다르다.
- `NULL` 값을 비교할 때는 `=` 대신 반드시 `IS NULL` 또는 `IS NOT NULL`을 사용해야 한다.
- MySQL에서 `NULL`은 정렬 시 가장 작은 값으로 취급되어 오름차순(`ASC`)에서는 맨 앞에 내림차순(`DESC`)에서는 맨 뒤에 위치한다.
- `ORDER BY (컬럼 IS NULL) DESC, 컬럼 DESC`와 같이 다중 정렬을 활용하면 `NULL` 값의 정렬 위치를 원하는 대로 제어할 수 있다.