5. UNION

#0.강의/2.데이터베이스로드맵/2.기본

UNION

우리는 지금까지 `JOIN`, 서브쿼리라는 강력한 도구들을 배웠다. 이 기술들의 공통점은 기존 테이블의 정보를 조합하거나 필터링해서, 우리가 원하는 형태의 '하나의 결과 집합(Result Set)'을 만들어내는 것이었다.

`JOIN`이 여러 테이블을 **옆으로(수평으로)** 붙여서 더 많은 정보를 가진 컬럼들을 만드는 기술이었다면, 지금부터 배울 `UNION`은 여러 개의 결과 집합을 **아래로(수직으로)** 이어 붙여서 더 많은 행을 가진 하나의 집합으로 만드는 기술이다.

`JOIN`이 정보를 풍성하게 만드는 기술이라면, `UNION`은 흩어진 집합들을 하나로 모으는 기술이라고 할 수 있다.

이 개념을 이해하기 위해, 오늘의 문제 상황을 살펴보자.
"우리 쇼핑몰은 현재 활동 중인 고객을 `users` 테이블에, 과거에 탈퇴한 고객을 `retired_users`라는 별도의 테이블에 보관하고 있다. 연말을 맞아 모든 고객(활동+탈퇴)에게 감사 이메일을 보내기 위해, 두 테이블에 흩어져 있는 이름과 이메일을 합쳐서 하나의 전체 목록을 만들어야 한다."

이 업무는 `JOIN`으로는 해결할 수 없다. 두 테이블은 서로 연결된 관계가 아니라, 구조는 비슷하지만 분리된 별개의 집합이기 때문이다.

실습 준비: `retired_users` 테이블 생성

이 문제를 실습하기 위해, 먼저 '탈퇴한 고객' 정보를 담을 `retired_users` 테이블을 임시로 만들어 보자. 일부러 현재 활동 고객인 '션'을 탈퇴 고객 명단에도 포함시켰다. 왜 그랬는지는 곧 알게 될 것이다.

- '션'은 `users` 테이블에도 포함되고 `retired_users` 테이블에도 포함된다. (비즈니스 로직에는 맞지 않지만 개발자의 실수로 두 곳에 모두 입력되었다고 가정하자.)

> **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.

```sql
-- 본 실습을 위한 탈퇴 고객 테이블 생성
DROP TABLE IF EXISTS retired_users;
CREATE TABLE retired_users (
    id BIGINT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    retired_date DATE NOT NULL
);

-- 탈퇴 고객 데이터 입력
INSERT INTO retired_users (id, name, email, retired_date) VALUES
(1, '션', 'sean@example.com', '2024-12-31'),
(7, '아이작 뉴턴', 'newton@example.com', '2025-01-10');
```

UNION의 개념과 사용법

이제 우리에겐 두 개의 고객 명단이 있다.

1. 활동 고객 명단 (`users` 테이블)

```sql
SELECT name, email FROM users;
```

name	email
션	sean@example.com
네이트	nate@example.com
세종대왕	sejong@example.com
이순신	sunsin@example.com
마리 퀴리	marie@example.com
레오나르도 다빈치	vinci@example.com

2. 탈퇴 고객 명단 (`retired_users` 테이블)

```
SELECT name, email FROM retired_users;
```

name	email
션	sean@example.com
아이작 뉴턴	newton@example.com

`UNION` 연산자는 이 두 개의 `SELECT` 문의 결과를 하나로 합쳐준다. 사용법은 아주 간단하다. 두 `SELECT` 문 사이에 `UNION` 키워드를 넣어주기만 하면 된다.

```
SELECT name, email FROM users
UNION
SELECT name, email FROM retired_users;
```

`UNION` 사용의 핵심 규칙

`UNION`을 사용할 때는 반드시 지켜야 할 중요한 규칙이 있다.

- `UNION`으로 연결되는 모든 `SELECT` 문은 **컬럼의 개수가 동일해야 한다.**
- 각 `SELECT` 문의 같은 위치에 있는 컬럼들은 서로 **호환 가능한 데이터 타입**이어야 한다. (예: 숫자 타입은 숫자 타입끼리, 문자 타입은 문자 타입끼리)
- 최종 결과의 컬럼 이름은 첫 번째 `SELECT` 문의 컬럼 이름을 따른다.

이제 `UNION` 쿼리의 실행 결과를 확인해 보자.

name	email
션	sean@example.com
네이트	nate@example.com
세종대왕	sejong@example.com
이순신	sunsin@example.com

마리 퀴리	marie@example.com
레오나르도 다빈치	vinci@example.com
아이작 뉴턴	newton@example.com

결과를 자세히 살펴보자. '션'은 `users` 테이블에도 있고 `retired_users` 테이블에도 있었다. 하지만 최종 결과 목록에는 **단 한 번만** 나타난다.

이것이 `UNION`의 특징이다. `UNION`**은 기본적으로 두 결과 집합을 합친 뒤, 완전히 중복되는 행은 자동으로 제거하여 고유한 값만 남긴다.**

이메일 목록을 만들 때 중복된 주소로 두 번 발송하는 실수를 원천적으로 방지해주니 매우 유용한 기능이다.

하지만 의도적으로 중복을 허용해야 하거나, 데이터에 중복이 없다는 것을 이미 알아서 굳이 중복 검사를 할 필요가 없는 경우도 있다. 바로 이럴 때 `UNION ALL`을 사용한다.

다음 시간에는 `UNION`과 `UNION ALL`의 차이점과 각각을 언제 사용해야 하는지, 그 성능 차이에 대해 알아보겠다.

UNION ALL

지난 시간에 우리는 `UNION`을 사용하여 활동 고객과 탈퇴 고객의 이메일 목록을 하나로 합치는 데 성공했다. 그 과정에서 `UNION`이 기본적으로 중복된 데이터를 알아서 제거해준다는 아주 중요한 특징을 발견했다. '션' 고객이 양쪽 테이블에 모두 있었지만, 최종 결과에는 한 번만 포함되었던 것을 기억할 것이다.

데이터의 중복을 제거해주는 것은 매우 편리한 기능이지만, 항상 우리에게 필요한 기능일까?
여기서 오늘의 문제 상황을 살펴보자.

"마케팅팀에서 두 종류의 고객에게 이벤트 안내 메일을 보내려고 한다. 첫 번째 그룹은 '전자기기' 카테고리의 상품을 구매한 이력이 있는 고객이고, 두 번째 그룹은 '서울'에 거주하는 고객이다. 두 그룹의 명단을 합쳐서 전체 발송 목록을 만들고 싶다."

여기서 중요한 질문이 생긴다. **'서울'에 살면서 '전자기기'를 구매한 고객은 두 그룹에 모두 속하게 되는데, 이 고객을 최**

종 목록에 한 번만 포함해야 할까? 아니면 중복을 허용해도 될까?

정답은 "비즈니스 요구사항에 따라 다르다" 이다. 하지만 이 선택에 따라 우리는 `UNION`과 `UNION ALL` 중 무엇을 쓸지 결정해야 한다.

`UNION`과 `UNION ALL`의 차이

`UNION`과 `UNION ALL`의 유일한 차이점은 '중복 처리' 여부다.

- `UNION`: 두 결과 집합을 합친 후, **중복된 행을 제거**한다.
- `UNION ALL`: 중복 제거 과정 없이, 두 결과 집합을 **그대로 모두 합친다**.

두 연산자의 차이를 쿼리 결과로 직접 확인해 보자.

`UNION` 사용 시 (중복 제거)

먼저, 두 고객 그룹을 `UNION`으로 합쳐보자.

```
-- 전자기기 구매 고객
SELECT u.name, u.email
FROM users u
INNER JOIN orders o ON u.user_id = o.user_id
INNER JOIN products p ON o.product_id = p.product_id
WHERE p.category = '전자기기'

UNION

-- 서울 거주 고객
SELECT name, email
FROM users
WHERE address LIKE '서울%';
```

- **전자기기 구매 고객**: 션, 마리 퀴리, 네이트, 네이트(중복), 이순신
- **서울 거주 고객**: 션, 세종대왕, 마리 퀴리
- **중복 고객**: 션, 네이트(중복), 마리 퀴리

`UNION`은 이 중복을 제거하고 고유한 목록만 보여준다.

[실행 결과]

name	email
션	sean@example.com
마리 퀴리	marie@example.com
네이트	nate@example.com
이순신	sunsin@example.com
세종대왕	sejong@example.com

총 5명의 고유한 고객 목록이 생성되었다. 중복된 '션'과 '네이트', '마리 퀴리'는 한 번씩만 포함되었다.

`UNION ALL` 사용 시 (중복 허용)

이제 똑같은 쿼리를 `UNION ALL`로 실행해 보자.

```
-- 전자기기 구매 고객
SELECT u.name, u.email
FROM users u
INNER JOIN orders o ON u.user_id = o.user_id
INNER JOIN products p ON o.product_id = p.product_id
WHERE p.category = '전자기기'

UNION ALL

-- 서울 거주 고객
SELECT name, email
FROM users
WHERE address LIKE '서울%';
```

`UNION ALL`은 각 `SELECT`문의 결과를 그대로 모두 이어 붙인다. '전자기기' 구매 내역은 총 5건(네이트 2건 포함)이고, '서울' 거주 고객은 3명이므로 총 8개의 행이 반환된다.

[실행 결과]

name	email
션	sean@example.com

마리 퀴리	marie@example.com
네이트	nate@example.com
네이트	nate@example.com
이순신	sunsin@example.com
션	sean@example.com
세종대왕	sejong@example.com
마리 퀴리	marie@example.com

결과를 보면 '션', '네이트', '마리 퀴리'가 중복되어 나타나는 것을 확인할 수 있다.

실무 가이드: 성능이 핵심이다

"그럼 언제 뭘 써야 하나요?" 라는 질문에 대한 답은 **성능**에 있다.

결론부터 말하면, `UNION ALL`이 `UNION`보다 훨씬 빠르다.

왜 그럴까? 데이터베이스의 입장에서 생각해 보자.

- `UNION`: 두 결과를 합친 뒤, 중복을 제거하기 위해 데이터베이스는 보이지 않는 곳에서 추가 작업을 해야 한다. 보통 전체 결과를 정렬(Sort)한 다음, 서로 인접한 행들을 비교하여 중복을 찾아내는 과정을 거친다. 데이터의 양이 수십만, 수백만 건이라면 이 정렬과 비교 작업은 엄청난 비용과 시간을 소모한다.
- `UNION ALL`: 이런 추가 작업이 전혀 없다. 그냥 첫 번째 `SELECT` 결과 아래에 두 번째 `SELECT` 결과를 가져다 붙이기만 하면 된다.

따라서 실무에서는 다음의 가이드라인을 따르는 것이 좋다.

1. **중복을 제거해야만 하는 명확한 요구사항이 있을 때만 `UNION`을 사용한다.**
   (예: 고유한 이메일 주소 목록, 고유한 고객 ID 목록 등)

2. **그 외의 모든 경우에는 `UNION ALL`을 우선적으로 사용한다.**
   - 두 결과 집합에 중복이 발생할 수 없다는 것을 명확히 아는 경우.
   - 중복이 발생해도 비즈니스 로직상 상관없는 경우.

가장 중요한 실무 팁은 이것이다. **"중복을 제거할 필요가 없다면, 항상 `UNION ALL`을 사용하자"** 불필요한 `UNION` 사

용은 쿼리를 느리게 만드는 주범이 될 수 있다.

UNION 정렬

`UNION` 또는 `UNION ALL`을 사용하여 여러 `SELECT` 문의 결과를 합칠 때, 최종 결과 집합에 대해 정렬(Sorting)을 적용할 수 있다. 이때 `ORDER BY` 절의 위치가 중요하다.

`ORDER BY` 절은 **전체** `UNION` **연산의 가장 마지막에 한 번만** 사용해야 한다. 만약 각 `SELECT` 문 안에 `ORDER BY`를 사용하면 에러가 발생하거나, 예상과 다른 결과가 나올 수 있다. 왜냐하면 `UNION`은 각 `SELECT` 문의 개별적인 정렬 순서가 아니라, 합쳐진 최종 결과 전체에 대한 순서를 결정해야 하기 때문이다.

예를 들어, 활동 고객과 탈퇴 고객의 명단을 합친 후, 이름(`name`)을 기준으로 오름차순 정렬하고 싶다고 가정해 보자.

```
SELECT name, email FROM users
UNION
SELECT name, email FROM retired_users
ORDER BY name; -- 최종 결과에 대한 정렬
```

[실행 결과]

name	email
네이트	nate@example.com
레오나르도 다빈치	vinci@example.com
마리 퀴리	marie@example.com
세종대왕	sejong@example.com
션	sean@example.com
아이작 뉴턴	newton@example.com
이순신	sunsin@example.com

`ORDER BY` 절을 `UNION` 연산자의 마지막에 배치함으로써, 합쳐진 모든 고객의 이름이 글자 순서대로 깔끔하게 정렬된 것을 확인할 수 있다.

UNION에 나오지 않는 필드를 사용한다면?

`UNION` 또는 `UNION ALL` 연산의 `ORDER BY` 절에서는 **첫 번째** `SELECT` **문의 컬럼 이름이나 해당 컬럼의 별칭 (Alias)만 사용할 수 있다.** 이는 `UNION` 결과 집합의 컬럼 이름이 첫 번째 `SELECT` 문을 따르기 때문이다.

```
SELECT name, email, created_at FROM users -- created_at
UNION ALL
SELECT name, email, retired_date FROM retired_users -- retired_date
```

[실행 결과]

name	email	created_at
션	sean@example.com	...
네이트	nate@example.com	...
세종대왕	sejong@example.com	..
이순신	sunsin@example.com	...
마리 퀴리	marie@example.com	...
레오나르도 다빈치	vinci@example.com	...
션	sean@example.com	...
아이작 뉴턴	newton@example.com	...

- 참고: 정렬 조건을 주지 않았기 때문에 정렬 결과는 달라질 수 있다.

첫 번째 SELECT 문은 `created_at` 두 번째 SELECT 문은 `retired_date`를 사용했다.
실행 결과를 보면 컬럼 이름에 `created_at`이 들어가 있는 것을 확인할 수 있다.

만약 첫 번째 `SELECT` 문에 없는 컬럼을 `ORDER BY` 절에서 사용하려고 하면 에러가 발생한다.

```
SELECT name, email, created_at FROM users
UNION ALL
SELECT name, email, retired_date FROM retired_users
```

```
ORDER BY retired_date; -- 첫 번째 SELECT 문에 없는 컬럼
```

[실행 결과]

```
Error Code: 1054. Unknown column 'retired_date' in 'order clause
```

별칭을 사용하기

다음의 `created_at`, `retired_date`과 같이 이름이 다른 컬럼이 있다면 별칭을 사용하는 것을 권장한다.
나중에 쿼리를 다시 보거나 다른 개발자가 볼 때, `created_at`이라는 이름만으로는 `retired_date`도 포함하여 정렬된다는 사실을 즉시 파악하기 어려울 수 있기 때문이다. `event_date`와 같은 중립적인 별칭은 해당 컬럼이 여러 종류의 날짜 정보를 담고 있음을 명확하게 보여준다.

```
SELECT name, email, created_at AS event_date FROM users
UNION ALL
SELECT name, email, retired_date AS event_date FROM retired_users
ORDER BY event_date DESC; -- 별칭을 사용하여 정렬
```

[실행 결과]

name	email	event_date
션	sean@example.com	...
네이트	nate@example.com	...
세종대왕	sejong@example.com	...
이순신	sunsin@example.com	...
마리 퀴리	marie@example.com	...
레오나르도 다빈치	vinci@example.com	...
아이작 뉴턴	newton@example.com	...
션	sean@example.com	...

- 참고: 데이터를 생성한 날짜가 다를 수 있기 때문에 정렬 결과를 달라질 수 있다.

지금까지 우리는 데이터를 조회하고(`SELECT`), 연결하고(`JOIN`), 합치는(`UNION`) 다양한 방법을 배웠다. 이제부터는 데이터에 **'조건부 로직'**을 부여하여, 조회 결과 자체를 동적으로 가공하는 방법을 배워볼 차례다. 예를 들어, 고객의 총 구매 금액에 따라 'VIP', 'Gold' 같은 등급을 매겨서 보여주고 싶다면 어떻게 해야 할까?

다음 섹션에서는 SQL의 `IF-THEN-ELSE`, 바로 `CASE` 문에 대해 알아보겠다.

문제와 풀이

문제1: 전체 고객 목록 조회하기

[문제]
우리 쇼핑몰의 모든 고객(활동 고객과 탈퇴 고객)의 이름과 이메일을 중복 없이 조회하여 하나의 목록으로 만들어라.

[실행 결과]

이름	이메일
션	sean@example.com
네이트	nate@example.com
세종대왕	sejong@example.com
이순신	sunsin@example.com
마리 퀴리	marie@example.com
레오나르도 다빈치	vinci@example.com
아이작 뉴턴	newton@example.com

[정답]

```
SELECT name AS 이름, email AS 이메일
```

```
FROM users
UNION
SELECT name, email
FROM retired_users;
```

문제2: 특별 이벤트 대상자 목록 만들기 (중복 포함)

[문제]

마케팅팀에서 두 그룹의 고객에게 이벤트를 진행하려고 한다.

1. '전자기기' 카테고리 상품을 한 번이라도 구매한 고객
2. 한 번의 주문으로 상품을 2개 이상 구매한 고객

두 그룹에 모두 해당하는 고객도 있을 수 있다. 성능을 고려하여, 두 그룹의 목록을 **중복 제거 없이** 모두 합쳐서 조회해라. 컬럼 별칭은 `고객명`, `이메일`로 지정한다.

[실행 결과]

고객명	이메일
선	sean@example.com
마리 퀴리	marie@example.com
네이트	nate@example.com
이순신	sunsin@example.com
선	sean@example.com
네이트	nate@example.com

[정답]

```
-- 전자기기 구매 고객
```

```sql
SELECT DISTINCT u.name AS 고객명, u.email AS 이메일
FROM users u
INNER JOIN orders o ON u.user_id = o.user_id
INNER JOIN products p ON o.product_id = p.product_id
WHERE p.category = '전자기기'

UNION ALL

-- 한 번에 2개 이상 구매한 고객
SELECT DISTINCT u.name AS 고객명, u.email AS 이메일
FROM users u
INNER JOIN orders o ON u.user_id = o.user_id
WHERE o.quantity > 1;
```

> **강의 내용 수정**
> 전자기기 구매 고객의 중복을 DISTINCT로 제거했다.

문제3: 회사 주요 이벤트 타임라인 만들기

[문제]
회사의 주요 이벤트 타임라인을 만들려고 한다. '고객 가입' 이벤트와 '상품 주문' 이벤트를 시간 순서대로 정렬하여 조회해라.

- `users` 테이블의 `created_at`은 '고객 가입' 이벤트로 간주한다.
- `orders` 테이블의 `order_date`는 '상품 주문' 이벤트로 간주한다.
- 결과는 `이벤트_날짜`, `이벤트_종류`, `상세_내용` 컬럼으로 구성한다.
- `상세_내용`에는 고객 가입 시 고객의 이름, 상품 주문 시 상품의 이름을 표시한다.
- 최신 이벤트가 가장 위에 오도록 내림차순으로 정렬해라.

[실행 결과]

이벤트 날짜	이벤트 종류	상세 내용
2025-08-19 20:33:32	고객 가입	션
2025-08-19 20:33:32	고객 가입	네이트

2025-08-19 20:33:32	고객 가입	세종대왕
2025-08-19 20:33:32	고객 가입	이순신
2025-08-19 20:33:32	고객 가입	마리 퀴리
2025-08-19 20:33:32	고객 가입	레오나르도 다빈치
2025-06-17 12:00:00	상품 주문	프리미엄 게이밍 마우스
2025-06-16 18:00:00	상품 주문	프리미엄 게이밍 마우스
2025-06-15 11:30:00	상품 주문	4K UHD 모니터
2025-06-12 09:00:00	상품 주문	관계형 데이터베이스 입문
2025-06-11 14:20:00	상품 주문	기계식 키보드
2025-06-10 10:05:00	상품 주문	관계형 데이터베이스 입문
2025-06-10 10:00:00	상품 주문	프리미엄 게이밍 마우스

- `고객 가입` 이벤트의 날짜는 쿼리 실행 시점의 `created_at` 값에 따라 달라질 수 있다.

[정답]

```sql
SELECT
    created_at AS 이벤트_날짜,
    '고객 가입' AS 이벤트_종류,
    name AS 상세_내용
FROM users

UNION ALL

SELECT
    o.order_date AS 이벤트_날짜,
    '상품 주문' AS 이벤트_종류,
    p.name AS 상세_내용
FROM orders o
INNER JOIN products p ON o.product_id = p.product_id

ORDER BY 이벤트_날짜 DESC;
```

문제4: 회사 전체 인명록 만들기

[문제]

회사의 모든 관련 인물(`users`의 고객과 `employees`의 직원)을 통합한 인명록을 만들어라.

- 결과에는 `이름`, `역할`('고객' 또는 '직원'), `이메일` 컬럼이 포함되어야 한다.
- 고객의 연락처는 `email` 컬럼을 사용한다.
- 직원의 연락처는 `name` 컬럼 뒤에 `'@my-shop.com'`을 붙여서 생성한다.
- 최종 결과는 이름(가나다/알파벳 순)으로 오름차순 정렬해라.

[실행 결과]

이름	역할	이메일
김회장	직원	김회장@my-shop.com
네이트	고객	nate@example.com
레오나르도 다빈치	고객	vinci@example.com
마리 퀴리	고객	marie@example.com
박사장	직원	박사장@my-shop.com
세종대왕	고객	sejong@example.com
션	고객	sean@example.com
이부장	직원	이부장@my-shop.com
이순신	고객	sunsin@example.com
정대리	직원	정대리@my-shop.com
최과장	직원	최과장@my-shop.com
홍사원	직원	홍사원@my-shop.com

[정답]

```
SELECT
    name AS 이름,
    '고객' AS 역할,
    email AS 이메일
FROM users

UNION ALL

SELECT
    name AS 이름,
    '직원' AS 역할,
    CONCAT(name, '@my-shop.com') AS 이메일
FROM employees

ORDER BY 이름;
```

정리

UNION

- `JOIN`이 테이블을 수평으로 붙여 컬럼을 늘리는 기술이라면 `UNION`은 여러 결과 집합을 수직으로 붙여 행을 늘리는 기술이다.
- 흩어져 있는 데이터를 하나의 집합으로 모으는 데 사용한다.
- `UNION`으로 연결하는 모든 `SELECT` 문은 컬럼 개수가 같고 같은 위치의 컬럼은 호환 가능한 데이터 타입이어야 한다.
- 최종 결과의 컬럼 이름은 첫 번째 `SELECT` 문의 컬럼 이름을 따른다.
- 두 결과 집합을 합친 뒤 중복되는 행은 자동으로 제거하여 고유한 값만 남긴다.

UNION ALL

- `UNION`과 `UNION ALL`의 유일한 차이점은 중복 처리 여부이다.
- `UNION`은 중복된 행을 제거하지만 `UNION ALL`은 중복 제거 없이 결과 집합을 그대로 모두 합친다.
- 성능은 `UNION ALL`이 `UNION`보다 훨씬 빠르다.
- `UNION`은 중복 제거를 위해 정렬과 비교 작업을 추가로 수행하기 때문이다.

- 중복을 명확히 제거해야 하는 경우가 아니라면 성능을 위해 항상 `UNION ALL`을 우선적으로 사용하는 것이 좋다.

UNION 정렬

- `UNION`으로 합친 최종 결과 집합을 정렬하려면 `ORDER BY` 절을 전체 쿼리의 가장 마지막에 한 번만 사용해야 한다.
- `ORDER BY` 절에서는 첫 번째 `SELECT` 문의 컬럼 이름이나 별칭만 사용할 수 있다.
- 첫 번째 `SELECT` 문에 없는 컬럼을 기준으로 정렬하면 오류가 발생한다.
- 서로 다른 이름의 컬럼을 합쳐 정렬할 때는 별칭을 사용하는 것을 권장한다.