6. CASE 문

#0.강의/2.데이터베이스로드맵/2.기본

CASE 문 기본1

우리는 지금까지 데이터를 있는 그대로 조회하거나(`SELECT`), 여러 테이블을 합치거나(`JOIN`, `UNION`), 특정 조건으로 걸러내는(`WHERE`) 방법을 배웠다. 즉, 데이터의 '구조'를 바꾸거나 '범위'를 한정하는 기술들이었다.

이제부터 배울 `CASE` **문**은 한 차원 다른 역할을 한다. `CASE` 문은 데이터 '자체'를 동적으로 가공하고 새로운 의미를 부여하는, 데이터에 멋진 옷을 입히는 기술이다. 마치 `IF-THEN-ELSE`처럼, 특정 조건에 따라 다른 값을 출력하게 만드는 SQL의 강력한 조건부 로직 도구다.

오늘의 문제 상황을 보자.
"상품 목록을 조회하는데, 그냥 가격만 보여주지 말고, 가격대에 따라 '고가', '중가', '저가'와 같은 알아보기 쉬운 등급을 옆에 함께 표시하고 싶다."

- 10만원 이상: '고가'
- 3만원 이상 10만원 미만: '중가'
- 3만원 미만: '저가'

물론 이 작업은 데이터베이스에서 원본 데이터를 모두 가져온 뒤, 자바나 파이썬 같은 애플리케이션 코드에서 `if-else` 문으로 처리할 수도 있다. 하지만 CASE 문을 사용하면 간단한 보고서나 데이터 분석을 할 때, 쿼리 하나만으로 원하는 최종 결과물을 바로 얻을 수 있어 매우 편리하다.

CASE 문의 두 가지 종류

`CASE` 문은 크게 두 가지 방식으로 나눌 수 있다. **단순** `CASE` **문(Simple CASE Expression)**과 **검색** `CASE` **문(Searched CASE Expression)**이다. 이름은 어렵지만, 실제 사용법은 매우 직관적이다.

단순 CASE 문 (Simple CASE Expression)

단순 `CASE` 문은 **특정 하나의 컬럼이나 표현식의 값에 따라 결과를 다르게 하고 싶을 때 사용한다.** 특정 값을 기준으로 "이 값이 A면 X, B면 Y, 그 외에는 Z"와 같이 명확하게 분기할 때 유용하다.

단순 CASE 문의 기본 문법

```
CASE 비교대상_컬럼_또는_표현식
    WHEN 값1 THEN 결과1
    WHEN 값2 THEN 결과2
    ...
    ELSE 그_외의_경우_결과
END
```

- `비교대상_컬럼_또는_표현식` : `WHEN` 절에서 비교할 대상이 되는 컬럼 또는 표현식이다.
- `WHEN 값 THEN 결과` : `비교대상_컬럼_또는_표현식`의 값이 `값`과 같을 경우 `결과`를 반환한다.
- `ELSE 결과` : 위에 명시된 `WHEN` 조건들 중 어느 것 하나도 참이 아닐 경우, `ELSE` 뒤의 결과를 반환한다. `ELSE`를 생략했는데 모든 `WHEN` 조건이 거짓이면 `NULL`이 반환된다.
- **실행 순서**: 단순 `CASE` 문도 위에서 아래 순서대로 조건을 평가하며, **가장 먼저 일치하는 `WHEN` 절을 만나는 순간 그 `THEN`의 결과를 반환하고 즉시 평가를 종료한다.**

단순 CASE 문 예제: 주문 상태(status)를 한글로 표시하기

우리 쇼핑몰의 `orders` 테이블에는 주문 상태를 `PENDING`, `COMPLETED`, `SHIPPED`, `CANCELLED`와 같은 영어로 저장하고 있다. 고객에게는 이 상태를 한글로 보여주고 싶다.

```
SELECT
    order_id,
    user_id,
    product_id,
    quantity,
    status,
    CASE status
        WHEN 'PENDING' THEN '주문 대기'
        WHEN 'COMPLETED' THEN '결제 완료'
        WHEN 'SHIPPED' THEN '배송'
        WHEN 'CANCELLED' THEN '주문 취소'
```

```
        ELSE '알 수 없음' -- 예상치 못한 상태 값 처리
    END AS status_korean
FROM
    orders;
```

이 쿼리는 `orders` 테이블의 각 행마다 `status` 컬럼의 값을 확인하여 해당하는 한글 상태 값을 `status_korean` 이라는 새로운 컬럼으로 반환한다.

[실행 결과]

order_id	user_id	product_id	quantity	status	status_korean
1	1	1	1	COMPLETED	결제 완료
2	1	4	2	COMPLETED	결제 완료
3	2	2	1	SHIPPED	배송
4	3	4	1	COMPLETED	결제 완료
5	4	3	1	PENDING	주문 대기
6	5	1	1	COMPLETED	결제 완료
7	2	1	2	SHIPPED	배송

단순 `CASE` 문은 이렇게 하나의 기준 값에 따라 명확하게 분류할 때 깔끔하고 가독성이 좋다.

검색 CASE 문 (Searched CASE Expression)

검색 `CASE` 문은 **단순 `CASE` 문처럼 하나의 특정 값을 비교하는 대신, 각 `WHEN` 절에 독립적인 조건식을 사용하여 복잡한 논리를 구현할 때 사용한다.** 앞서 만난 문제 상황처럼 '가격이 얼마 이상', '날짜가 언제 이전'과 같은 범위 조건이나 여러 컬럼을 조합한 복합적인 조건이 필요할 때 주로 사용된다.

검색 CASE 문의 기본 문법

```
CASE
    WHEN 조건1 THEN 결과1
    WHEN 조건2 THEN 결과2
```

```
    ...
    ELSE 그_외의_경우_결과
END
```

- `WHEN 조건 THEN 결과`: `WHEN` 뒤의 조건이 참(true)일 경우, `THEN` 뒤의 결과를 반환한다.
  - 여기서 조건은 `price >= 100000`, `category = '도서'` 등 다양한 비교 연산자와 논리 연산자(`AND`, `OR`, `NOT`)를 포함할 수 있다.
- `ELSE 결과`: 위에 명시된 `WHEN` 조건들 중 어느 것 하나도 참이 아닐 경우, `ELSE` 뒤의 결과를 반환한다. `ELSE`를 생략했는데 모든 `WHEN` 조건이 거짓이면 `NULL`이 반환된다.
- **실행 순서**: 검색 `CASE` 문도 위에서 아래 순서대로 조건을 평가하며, **가장 먼저 참이 되는 `WHEN` 절을 만나는 순간 그 `THEN`의 결과를 반환하고 즉시 평가를 종료한다.** 이 순서가 매우 중요하다.

검색 CASE 문 예제: 상품 가격에 따라 등급 표시하기

이제 강의 시작 시점에 제시했던 문제, 즉 상품 가격에 따라 '고가', '중가', '저가' 등급을 표시하는 문제를 검색 `CASE` 문으로 해결해 보자.

```sql
SELECT
    name,
    price,
    CASE
        WHEN price >= 100000 THEN '고가'
        WHEN price >= 30000 THEN '중가'
        ELSE '저가'
    END AS price_label
FROM
    products;
```

여기서 `AS price_label`을 사용해, `CASE` 문 전체가 만들어내는 결과에 `price_label`이라는 새로운 컬럼명을 붙여주었다.

이 쿼리는 `products` 테이블의 각 행마다 다음의 로직을 수행한다.

1. 가격이 10만원 이상인가? 맞으면 '고가'를 반환하고 `CASE` 문을 끝낸다.
2. **(1번이 아니라면)** 가격이 3만원 이상인가? 맞으면 '중가'를 반환하고 끝낸다.
3. **(1, 2번이 모두 아니라면)** `ELSE`에 따라 '저가'를 반환한다.

[실행 결과]

name	price	price_label
프리미엄 게이밍 마우스	75000	중가
기계식 키보드	120000	고가
4K UHD 모니터	350000	고가
관계형 데이터베이스 입문	28000	저가
고급 가죽 지갑	150000	고가
스마트 워치	280000	고가

보는 것과 같이, 원본 데이터는 전혀 건드리지 않고, 조회 결과에만 우리가 정의한 비즈니스 로직에 따라 새로운 값을 동적으로 생성하여 보여주었다.

CASE 문 기본2

검색 CASE 문 사용 시 주의사항: WHEN 절의 순서

앞서 `CASE` 문은 위에서 아래로 순차적으로 평가하며, **가장 먼저 참이 되는 조건을 만나는 순간 실행을 멈춘다**고 강조했다. 이 점이 검색 `CASE` 문에서는 특히 중요하다. 만약 조건을 잘못 배치하면 예상과 다른 결과가 나올 수 있다.

만약 위 쿼리에서 `WHEN price >= 30000` 조건을 `WHEN price >= 100000` 조건보다 먼저 배치했다면 어떻게 될까?

```
-- 잘못된 순서의 예 (의도와 다른 결과)
SELECT
    name,
    price,
    CASE
        WHEN price >= 30000 THEN '중가' -- 이 조건이 먼저 평가된다
        WHEN price >= 100000 THEN '고가'
        ELSE '저가'
    END AS price_label
```

```
FROM
    products;
```

[실행 결과]

name	price	price_label
프리미엄 게이밍 마우스	75000	중가
기계식 키보드	120000	중가
4K UHD 모니터	350000	중가
관계형 데이터베이스 입문	28000	저가
고급 가죽 지갑	150000	중가
스마트 워치	280000	중가

- `프리미엄 게이밍 마우스` (75000원): 75000원은 30000원 이상이므로 '중가'로 분류된다. (올바름)
- `기계식 키보드` (120000원): 120000원은 30000원 이상이므로 **'중가'로 분류된다.** (틀림! '고가'여야 함)

이처럼 `WHEN price >= 100000`인 상품도 `price >= 30000` 조건을 먼저 만족시켜 '중가'로 잘못 분류될 것이다. 따라서 검색 `CASE` 문을 사용할 때는 조건의 순서를 신중하게 고려해야 한다. **더 포괄적인(범위가 넓은) 조건보다는 더 구체적인(범위가 좁은) 조건을 먼저 배치하는 것이 일반적이다.**

CASE 문과 사용 위치

`CASE` 문은 `SELECT` 절 외에도 `ORDER BY`, `GROUP BY`, `WHERE` 절 등 다양한 SQL 구문과 함께 사용될 수 있다.

예를 들어, 상품을 '고가', '중가', '저가' 순서로 정렬하고 싶다면 `ORDER BY` 절에 `CASE` 문을 사용할 수 있다.

```
SELECT
    name,
    price,
    CASE
        WHEN price >= 100000 THEN '고가'
        WHEN price >= 30000 THEN '중가'
        ELSE '저가'
    END AS price_label
FROM
```

```
    products
ORDER BY
    CASE
        WHEN price >= 100000 THEN 1   -- 고가: 1
        WHEN price >= 30000 THEN 2    -- 중가: 2
        ELSE 3                        -- 저가: 3
    END ASC, -- 숫자가 작은 순서대로 정렬
    price DESC; -- 같은 등급 내에서는 가격 내림차순
```

[실행 결과]

name	price	price_label
4K UHD 모니터	350000	고가
스마트 워치	280000	고가
고급 가죽 지갑	150000	고가
기계식 키보드	120000	고가
프리미엄 게이밍 마우스	75000	중가
관계형 데이터베이스 입문	28000	저가

보는 것처럼, `price_label` 컬럼의 '고가', '중가', '저가' 문자열 순서가 아닌, 우리가 `CASE` 문에서 지정한 1, 2, 3의 숫자 순서대로 정렬된 것을 확인할 수 있다. 이렇게 `CASE` 문은 데이터의 표현뿐만 아니라 정렬, 그룹화 등 다양한 로직에 활용할 수 있다.

이것이 `CASE` 문의 기본이다. 하지만 `CASE` 문의 진정한 힘은 다른 SQL 구문, 특히 집계 함수나 `GROUP BY`와 결합될 때 발휘된다.

단순히 등급을 표시하는 것을 넘어, '고가', '중가', '저가' 상품이 각각 몇 개씩 있는지 집계하려면 어떻게 해야 할까? 다음 시간에는 `CASE` 문을 활용하여 데이터를 분류하고 그룹핑하는 방법에 대해 알아보겠다.

CASE 문 - 그룹핑

데이터 분류 및 그룹핑

지난 시간에는 `CASE` 문의 기본 문법을 배우고, 각 상품에 '고가', '중가', '저가'라는 라벨을 붙여보는 실습을 했다. `CASE` 문이 각 행에 대해 새로운 값을 동적으로 만들어내는 것을 확인했다.

하지만 `CASE` 문의 진정한 힘은, 이렇게 동적으로 만들어낸 값을 **다른 SQL 구문과 결합**할 때 드러난다. 오늘은 `CASE` 문과 `GROUP BY`를 함께 사용하여, 데이터를 우리가 원하는 기준으로 분류하고, 분류된 그룹별로 통계를 내는 실용적인 기술을 배워보겠다.

오늘의 문제 상황이다.
"고객들을 출생 연대에 따라 '1990년대생', '1980년대생', '그 이전 출생'으로 분류하고, 각 그룹에 고객이 총 몇 명씩 있는지 알고 싶다."

이 문제를 해결하기 위한 전략은 두 단계로 나뉜다.

1. **분류(Classification)**: `CASE` 문을 사용해 각 고객에게 '1990년대생', '1980년대생', '그 이전 출생' 중 하나의 라벨을 붙여준다.
2. **집계(Aggregation)**: 1단계에서 만들어진 라벨을 기준으로 `GROUP BY` 하고, `COUNT()` 함수를 사용해 각 라벨(그룹)에 속한 고객 수를 센다.

단계별 쿼리 작성

먼저, 1단계인 '분류' 작업부터 쿼리로 작성해 보자. `users` 테이블의 `birth_date` 컬럼에서 `YEAR()` 함수로 연도만 추출하여 조건을 만든다.

1단계: `CASE` 문으로 각 고객 분류하기
아직 `GROUP BY`는 하지 않고, 각 고객이 어떤 라벨을 받게 되는지부터 눈으로 확인한다.

```sql
SELECT
    name,
    birth_date,
    CASE
        WHEN YEAR(birth_date) >= 1990 THEN '1990년대생'
        WHEN YEAR(birth_date) >= 1980 THEN '1980년대생'
        ELSE '그 이전 출생'
    END AS birth_decade
FROM
    users;
```

[실행 결과]

name	birth_date	birth_decade
션	1990-01-15	1990년대생
네이트	1988-05-22	1980년대생
세종대왕	1397-05-15	그 이전 출생
이순신	1545-04-28	그 이전 출생
마리 퀴리	1867-11-07	그 이전 출생
레오나르도 다빈치	1452-04-15	그 이전 출생

이렇게 `CASE` 문이 만들어낸 새로운 가상 컬럼 `birth_decade`가 성공적으로 생성된 것을 볼 수 있다.

2단계: `GROUP BY`로 그룹화하고 `COUNT`로 집계하기

이제 2단계 전략을 실행할 차례다. 1단계에서 만든 저 가상 컬럼 `birth_decade`를 `GROUP BY`의 기준으로 삼으면 된다.

```
SELECT
    CASE
        WHEN YEAR(birth_date) >= 1990 THEN '1990년대생'
        WHEN YEAR(birth_date) >= 1980 THEN '1980년대생'
        ELSE '그 이전 출생'
    END AS birth_decade,
    COUNT(*) AS customer_count
FROM
    users
GROUP BY
    CASE
        WHEN YEAR(birth_date) >= 1990 THEN '1990년대생'
        WHEN YEAR(birth_date) >= 1980 THEN '1980년대생'
        ELSE '그 이전 출생'
    END;
```

[실행 결과]

birth_decade	customer_count
1990년대생	1
1980년대생	1
그 이전 출생	4

SQL 표준의 논리적 실행 순서에 따르면 `GROUP BY` 절이 `SELECT` 절보다 먼저 처리된다. 따라서 원칙적으로는 `SELECT` 절에서 정의한 별칭(`birth_decade`)을 `GROUP BY` 절에서 사용할 수 없다. 하지만 MySQL을 포함한 최신 버전의 많은 데이터베이스들은 사용자 편의를 위해 이러한 별칭 사용을 예외적으로 허용한다.

```
SELECT
    CASE
        WHEN YEAR(birth_date) >= 1990 THEN '1990년대생'
        WHEN YEAR(birth_date) >= 1980 THEN '1980년대생'
        ELSE '그 이전 출생'
    END AS birth_decade,
    COUNT(*) AS customer_count
FROM
    users
GROUP BY
    birth_decade;
```

`GROUP BY` 절에 `CASE` 문 전체를 다시 쓸 필요 없이, `SELECT` 절에서 `AS`로 지정한 별칭(`birth_decade`)을 바로 사용했다.
이는 쿼리를 훨씬 간결하고 읽기 쉽게 만들어준다.

드디어 우리가 원했던 최종 보고서, '출생 연대별 고객 수'를 완벽하게 만들어냈다.

이처럼 `CASE` 문을 이용해 기존에 없던 새로운 분류 기준을 동적으로 생성하고, 이를 `GROUP BY`와 결합하는 패턴은 실무 데이터 분석과 리포팅에서 정말로 흔하게 사용되는 핵심 기술 중 하나다.

`CASE` 문은 이렇게 그룹의 '기준'을 만드는 역할도 하지만, 집계 함수인 `COUNT`나 `SUM`의 '내부'로 들어가서 특정 조건에 맞는 값만 골라서 집계하는 훨씬 더 정교한 역할도 수행할 수 있다.

다음 시간에는 집계 함수 안에서 `CASE` 문을 활용하는 고급 기법에 대해 알아보겠다.

CASE 문 - 조건부 집계1

지난 시간에는 `CASE` 문으로 새로운 분류 기준('연령대')을 만들고, 이 기준을 `GROUP BY` 절에 사용하여 그룹별 통계를 내는 법을 배웠다. `CASE` 문이 집계 함수의 '바깥'에서 그룹의 기준을 만드는 역할을 했다.

오늘은 여기서 한 걸음 더 나아가, `CASE` 문이 집계 함수(`SUM`, `COUNT` 등)의 '안'으로 들어가는, 훨씬 더 강력한 활용법을 배워본다. 이 기법을 **조건부 집계(Conditional Aggregation)**라고 한다.

오늘의 문제 상황은 다음과 같다. 엑셀의 '피벗 테이블' 기능과 유사한 보고서를 SQL로 직접 만들어보는 것이다.
"하나의 쿼리로, 전체 주문 건수와 함께 '결제 완료(COMPLETED)', '배송(SHIPPED)', '주문 대기(PENDING)' 상태의 주문이 각각 몇 건인지 별도의 컬럼으로 나누어 보고 싶다."

정리하면 다음과 같은 보고서를 만들고 싶다는 의미다.

total_orders	completed_count	shipped_count	pending_count
7	4	2	1

> **피벗 테이블(Pivot Table)**
> "피벗 테이블(Pivot Table)"이라는 이름은 데이터를 다양한 관점에서 "회전(pivot)" 시켜 분석할 수 있는 기능 때문에 붙여진 이름이다.

이 문제를 해결하기 위해 단계별로 접근해보자.

먼저 주문 테이블의 데이터를 확인해보자.

```sql
SELECT order_id, status FROM orders;
```

order_id	status

1	COMPLETED
2	COMPLETED
3	SHIPPED
4	COMPLETED
5	PENDING
6	COMPLETED
7	SHIPPED

이 테이블을 각 주문의 상태별로 그룹핑 해보자.

```
SELECT status, COUNT(*) FROM orders
GROUP BY status;
```

status	COUNT(*)
COMPLETED	4
SHIPPED	2
PENDING	1

'결제 완료(COMPLETED)', '배송(SHIPPED)', '주문 대기(PENDING)' 상태의 주문이 각각 몇 건인지 구했다.
여기까지는 해결하기 어렵지 않다. 그런데 전체 GROUP BY를 사용했기 때문에 각 그룹별 통계만 구할 수 있다. 전체 합을 구하려면 어떻게 해야할까?

전체 합 구하기

전체 합을 구하려면 `GROUP BY` 절을 사용하지 않고 `COUNT(*)`와 같은 집계 함수를 사용하면 된다.

```
SELECT COUNT(*) AS total_orders FROM orders;
```

[실행 결과]

total_orders

7

이제 둘을 UNION으로 합치는 방법도 있고, 서브 쿼리를 사용하는 방법도 있다.

UNION으로 합치는 방법

`UNION`을 사용하면 여러 `SELECT` 문의 결과를 하나로 합칠 수 있다. 이때 각 `SELECT` 문의 컬럼 수가 같아야 하며, 컬럼의 데이터 타입도 호환되어야 한다.

```
SELECT 'Total' AS category, COUNT(*) AS count FROM orders
UNION ALL
SELECT status AS category, COUNT(*) AS count FROM orders GROUP BY status;
```

[실행 결과]

category	count
Total	7
COMPLETED	4
SHIPPED	2
PENDING	1

이 방법은 각 상태별 통계와 전체 합계를 하나의 결과 집합으로 보여주지만, 우리가 원하는 "각 상태별 카운트가 **별도의 컬럼**으로 나오는" 보고서 형식과는 차이가 있다. 이 모양을 옆으로 돌려서 피벗 테이블의 형태가 되어야 한다.

서브 쿼리를 사용하는 방법

각 상태별 카운트를 서브 쿼리로 미리 구한 다음, 메인 쿼리에서 이 값들을 조인하거나 활용하는 방법이 있다.

```
SELECT
    (SELECT COUNT(*) FROM orders) AS total_orders,
    (SELECT COUNT(*) FROM orders WHERE status = 'COMPLETED') AS
completed_count,
    (SELECT COUNT(*) FROM orders WHERE status = 'SHIPPED') AS shipped_count,
    (SELECT COUNT(*) FROM orders WHERE status = 'PENDING') AS pending_count;
```

[실행 결과]

total_orders	completed_count	shipped_count	pending_count
7	4	2	1

이 방법은 우리가 목표로 하는 보고서 형식과 같은 결과를 보여준다. 각 카운트가 별도의 컬럼으로 잘 분리되어 있다. 이 방법은 우리가 원하는 결과를 주지만, 각 값을 얻기 위해 `orders` 테이블을 총 4번이나 읽어오는(`FROM orders`) 심각한 성능 문제를 가지고 있다. 데이터가 많아질수록 매우 비효율적이다.

바로 이럴 때 `CASE` 문을 집계 함수 안에 넣는 조건부 집계 기법이 필요하다. 이 방법은 단일 쿼리 내에서 효율적으로 조건별 집계를 수행할 수 있게 한다.

CASE 문 - 조건부 집계2

`CASE`를 품은 집계 함수

이 문제를 해결하는 핵심 아이디어는, 집계 함수의 인자로 `CASE` 문을 넣어 '조건에 맞을 때만 세거나 더하게' 만드는 것이다. 두 가지 대표적인 패턴이 있다.

패턴 1: `COUNT(CASE ...)`

`COUNT` 함수는 `NULL`이 아닌 모든 값을 센다는 특징을 이용한다.

```
COUNT(CASE WHEN status = 'COMPLETED' THEN 1 END)
```

- `status`가 'COMPLETED'이면 `CASE` 문은 숫자 `1`을 반환한다.
- 그 외의 경우(`ELSE`가 없으므로), `CASE` 문은 `NULL`을 반환한다.
- 결과적으로 `COUNT` 함수는 `status`가 'COMPLETED'인 행의 개수만 세게 된다.

패턴 2: `SUM(CASE ...)`

`SUM` 함수는 숫자들의 합계를 구한다.

```
SUM(CASE WHEN status = 'COMPLETED' THEN 1 ELSE 0 END)
```

- `status`가 'COMPLETED'이면 `CASE` 문은 `1`을, 그 외에는 `0`을 반환한다.

- `SUM` 함수가 이 1과 0들을 모두 더하면, 그 합계는 결국 'COMPLETED' 상태인 주문의 총 개수가 된다.

두 패턴 모두 동일한 결과를 내며, 실무에서 널리 쓰인다. 여기서는 각 상태의 합계를 구한다는 의미를 더 명확히 보여주는 `SUM` 패턴을 사용해 보겠다.

실습 1: 전체 주문 상태 요약하기

먼저 `orders` 테이블 전체를 대상으로, 각 상태별 주문 건수를 집계해 보자. `GROUP BY`는 아직 필요 없다.

```
SELECT
    COUNT(*) AS total_orders,
    SUM(CASE WHEN status = 'COMPLETED' THEN 1 ELSE 0 END) AS completed_count,
    SUM(CASE WHEN status = 'SHIPPED' THEN 1 ELSE 0 END) AS shipped_count,
    SUM(CASE WHEN status = 'PENDING' THEN 1 ELSE 0 END) AS pending_count
FROM
    orders;
```

[실행 결과]

total_orders	completed_count	shipped_count	pending_count
7	4	2	1

단 하나의 쿼리로 행으로 흩어져 있던 `status` 정보를 열로 변환하여, 한눈에 파악하기 쉬운 피벗(Pivot) 보고서를 만들어냈다.

실습 2: `GROUP BY`와 함께 사용하기 (피벗 테이블)

조건부 집계는 `GROUP BY`와 함께 사용할 때 그 진정한 힘을 발휘한다.
이번에는 한 단계 더 나아가 **"상품 카테고리별로, 상태별 주문 건수를 집계하라"** 는 보고서를 만들어 보자.

이를 위해서는 `orders` 테이블과 `products` 테이블을 `JOIN`하여 `category` 정보를 가져온 뒤, `p.category`를 기준으로 `GROUP BY` 해주면 된다.

```
SELECT
    p.category,
```

```
    COUNT(*) AS total_orders,
    SUM(CASE WHEN o.status = 'COMPLETED' THEN 1 ELSE 0 END) AS
completed_count,
    SUM(CASE WHEN o.status = 'SHIPPED' THEN 1 ELSE 0 END) AS shipped_count,
    SUM(CASE WHEN o.status = 'PENDING' THEN 1 ELSE 0 END) AS pending_count
FROM
    orders o
JOIN
    products p ON o.product_id = p.product_id
GROUP BY
    p.category;
```

[실행 결과]

category	total_orders	completed_count	shipped_count	pending_count
전자기기	5	2	2	1
도서	2	2	0	0

이제 우리는 카테고리별로 주문 현황을 훨씬 더 상세하고 입체적으로 분석할 수 있게 되었다. '전자기기'는 주문은 많지만 배송 상태이거나 대기중인 건이 많은 반면, '도서'는 주문 건수는 적지만 모두 처리가 완료되었다는 인사이트를 단번에 얻을 수 있다.

이처럼 `CASE` 문을 집계 함수 내부에 사용하는 조건부 집계 기법은, 데이터를 단순히 요약하는 것을 넘어 원하는 형태로 재구조화하고 분석하는 데 필요한 도구다.

방금 우리가 만든 '카테고리별 주문 현황' 쿼리는 매우 유용해서, 재무팀이나 마케팅팀에서 매일같이 필요로 할 수 있다. 하지만 이 복잡한 쿼리를 매번 다시 작성하거나 어딘가에 저장해 둔 것을 복사-붙여넣기 하는 것은 번거롭고 실수할 가능성도 높다.

이 복잡한 쿼리를 마치 하나의 간단한 테이블처럼 데이터베이스에 저장해두고, 필요할 때마다 `SELECT * FROM daily_report;` 처럼 쉽게 불러와 쓸 수는 없을까?

다음 섹션에서는 이 문제를 해결해 줄 마법 같은 가상 테이블, **뷰(VIEW)**에 대해 알아보겠다.

문제와 풀이

문제1: 상품 카테고리 영문으로 표시하기

[문제]

`products` 테이블에서 상품의 `category` 컬럼 값을 영문으로 변경하여 조회해라. '전자기기'는 'Electronics', '도서'는 'Books', '패션'은 'Fashion'으로 표시하고, `category_english` 라는 별칭을 사용해라. 상품 이름, 원래 카테고리, 그리고 영문 카테고리를 함께 출력해야 한다.

[실행 결과]

name	category	category_english
프리미엄 게이밍 마우스	전자기기	Electronics
기계식 키보드	전자기기	Electronics
4K UHD 모니터	전자기기	Electronics
관계형 데이터베이스 입문	도서	Books
고급 가죽 지갑	패션	Fashion
스마트 워치	전자기기	Electronics

[정답]

```
SELECT
    name,
    category,
    CASE category
        WHEN '전자기기' THEN 'Electronics'
        WHEN '도서' THEN 'Books'
        WHEN '패션' THEN 'Fashion'
        ELSE 'Etc'
    END AS category_english
FROM
```

```
    products;
```

문제2: 주문 수량에 따른 분류 및 정렬

[문제]

`orders` 테이블의 주문들을 수량(`quantity`)에 따라 분류하고 싶다. 수량이 2개 이상이면 '다량 주문', 1개이면 '단일 주문'으로 표시하는 `order_type` 컬럼을 만들어라. 주문 ID, 수량, 그리고 이 새로운 `order_type` 컬럼을 조회하되, '다량 주문'이 '단일 주문'보다 먼저 나오도록 정렬해라.

[실행 결과]

order_id	quantity	order_type
2	2	다량 주문
7	2	다량 주문
1	1	단일 주문
3	1	단일 주문
4	1	단일 주문
5	1	단일 주문
6	1	단일 주문

[정답]

```
SELECT
    order_id,
    quantity,
    CASE
        WHEN quantity >= 2 THEN '다량 주문'
        ELSE '단일 주문'
    END AS order_type
```

```
FROM
    orders
ORDER BY
    CASE
        WHEN quantity >= 2 THEN 1
        ELSE 2
    END ASC;
```

문제3: 재고 수준별 상품 수 집계하기

[문제]

`products` 테이블의 상품들을 재고 수량(`stock_quantity`)에 따라 그룹화하여 각 그룹에 속한 상품의 수를 세어 보아라.

- 재고가 50개 이상이면 '재고 충분'
- 20개 이상 50개 미만이면 '재고 보통'
- 20개 미만이면 '재고 부족'

`stock_level`과 `product_count` 라는 컬럼명으로 결과를 출력해야 한다.

[실행 결과]

stock_level	product_count
재고 충분	2
재고 보통	3
재고 부족	1

[정답]

```
SELECT
    CASE
```

```
        WHEN stock_quantity >= 50 THEN '재고 충분'
        WHEN stock_quantity >= 20 THEN '재고 보통'
        ELSE '재고 부족'
    END AS stock_level,
    COUNT(*) AS product_count
FROM
    products
GROUP BY
    stock_level;
```

문제4: 사용자별 카테고리 주문 건수 피벗 테이블 만들기

[문제]

각 사용자(`users`)가 카테고리별로 몇 건의 주문을 했는지 피벗 테이블 형태로 조회해라. 사용자 이름, 총 주문 건수, '전자기기' 주문 건수, '도서' 주문 건수를 각각 `user_name`, `total_orders`, `electronics_orders`, `book_orders`, `fashion_orders` 컬럼으로 출력해야 한다. 한 번도 주문하지 않은 고객도 결과에 포함해야 한다.

[실행 결과]

user_name	total_orders	electronics_orders	book_orders	fashion_orders
션	2	1	1	0
네이트	2	2	0	0
세종대왕	1	0	1	0
이순신	1	1	0	0
마리 퀴리	1	1	0	0
레오나르도 다빈치	0	0	0	0

[정답]

```
SELECT
```

```
    u.name AS user_name,
    COUNT(o.order_id) AS total_orders,
    SUM(CASE WHEN p.category = '전자기기' THEN 1 ELSE 0 END) AS
electronics_orders,
    SUM(CASE WHEN p.category = '도서' THEN 1 ELSE 0 END) AS book_orders,
    SUM(CASE WHEN p.category = '패션' THEN 1 ELSE 0 END) AS fashion_orders
FROM
    users u
LEFT JOIN
    orders o ON u.user_id = o.user_id
LEFT JOIN
    products p ON o.product_id = p.product_id
GROUP BY
    u.name;
```

정리

CASE 문 기본1

- `CASE` 문은 특정 조건에 따라 데이터를 동적으로 가공하고 새로운 의미를 부여하는 SQL의 조건부 로직 도구이다.
- `CASE` 문은 **단순 CASE 문**과 **검색 CASE 문**, 두 가지 종류로 나뉜다.
- **단순 CASE 문**은 특정 컬럼의 값이 A면 X, B면 Y와 같이 명확하게 정해진 값을 비교할 때 사용한다.
- **검색 CASE 문**은 `WHEN` 절마다 독립적인 조건식을 사용하여 가격 범위나 복합적인 조건을 처리할 때 사용한다.
- `CASE` 문은 위에서 아래로 순차적으로 조건을 평가하며, 가장 먼저 참이 되는 조건의 결과를 반환하고 즉시 평가를 종료한다.

CASE 문 기본2

- 검색 `CASE` 문을 사용할 때는 `WHEN` 절의 순서가 매우 중요하며, 잘못 배치하면 의도와 다른 결과가 나올 수 있다.
- 일반적으로 범위가 좁고 구체적인 조건을 더 넓고 포괄적인 조건보다 먼저 배치해야 한다.
- `CASE` 문은 `SELECT` 절뿐만 아니라 `ORDER BY`, `GROUP BY`, `WHERE` 절에서도 사용되어 데이터의 표현, 정렬, 그룹화 등 다양한 로직에 활용할 수 있다.

CASE 문 - 그룹핑

- `CASE` 문을 `GROUP BY` 절과 함께 사용하면, 기존에 없던 새로운 기준으로 데이터를 분류하고 그룹별 통계를 낼

수 있다.

- 쿼리 작성은 2단계로 이루어진다.
  - 1단계 (분류): `CASE` 문으로 각 행에 원하는 분류 라벨을 붙인다.
  - 2단계 (집계): 1단계에서 만든 라벨을 기준으로 `GROUP BY` 하고, `COUNT()` 같은 집계 함수로 그룹별 통계를 구한다.
- `SELECT` 절에서 `CASE` 문에 부여한 별칭을 `GROUP BY` 절에서 사용하여 쿼리를 간결하게 만들 수 있다.

CASE 문 - 조건부 집계1

- **조건부 집계**는 `CASE` 문을 `SUM`이나 `COUNT` 같은 집계 함수 내부에 넣어 사용하는 강력한 기법이다.
- 행으로 표현된 데이터를 열로 변환하는 피벗(Pivot) 형태의 보고서를 만들 때 유용하다.
- 각 조건별로 서브쿼리를 여러 번 사용하면 성능이 저하되므로, 조건부 집계는 테이블을 한 번만 읽어 효율적으로 처리하는 방법이다.

CASE 문 - 조건부 집계2

- 조건부 집계는 `COUNT(CASE ...)` 또는 `SUM(CASE ...)` 패턴을 주로 사용한다.
  - `COUNT(CASE WHEN 조건 THEN 1 END)` : 조건이 참일 때만 1을 반환하고, 나머지는 NULL을 반환하여 `COUNT`가 NULL을 제외하고 세도록 한다.
  - `SUM(CASE WHEN 조건 THEN 1 ELSE 0 END)` : 조건이 참이면 1, 아니면 0을 반환하여 합계를 구함으로써 개수를 센다.
- 조건부 집계를 `GROUP BY`와 함께 사용하면, '카테고리별, 상태별 주문 건수'처럼 다차원적인 분석 보고서를 효율적으로 생성할 수 있다.