- **GROUP BY/ AGGREGATE** 함수

    - **SUM()**:특정 컬럼의 합계를 구할 때 사용
    - **COUNT()**: 행의 개수를 셀 때 사용
    - **AVG()**: 특정 컬럼의 평균값을 구할 때 사용
    - **MIN(),MAX()**: 최솟값과 최댓값을 구할 때 사용

```sql
SELECT category,SUM(price) AS total_sales
FROM orders
GROUP BY category;
```

- **JOIN**

    - **INNER JOIN**: 두 테이블에 모두 존재하는 데이터만을 결합
    - **LEFT JOIN**: 왼쪽 테이블의 모든 데이터와 오른쪽 테이블의 일치하는 데이터만 결합
    - **RIGHT JOIN**: 오른쪽 테이블의 모든 데이터와 왼쪽 테이블의 일치하는 데이터만 결합
    - **OUTER JOIN**: 결합되는 테이블의 한쪽 또는 양쪽에 데이터가 없는 경우에도 결과에 포함

```sql
SELECT o.order_id, c.customer_name,o.totla_amount
FROM orders o
INNER JOIN customers c ON o.customer_id = c.customer_id;
```

```sql
SELECT o.order_id, c.customer_name, o.total_amount
FROM orders o
LEFT JOIN customers c ON o.customer_id = c.customer_id;
```

```sql
SELECT o.order_id, c.customer_name, o.total_amount
FROM orders o
RIGHT JOIN customers c ON o.customer_id = c.customer_id;
```

```sql
SELECT o.order_id, c.customer_name, o.total_amount
FROM orders o
FULL OUTER JOIN customers c ON o.customer_id = c.customer_id;
```

- **WITH 및 CTE(Common Table Expression)**

    - CTE: CTE는 복잡한 쿼리를 더 간결하고 가독성 있게 작성하기 위한 방법으로, 쿼리 안에 임시로 정의한 이름 있는 결과 집합입니다. CTE는 주로 `WITh` 구문과 함께 사용되며, 재사용 가능한 임시 결과를 만들고 그 결과를 메인 쿼리에서 참조할 수 있게 해줍니다.
    - `WITh` 구문:
      `WITH` 구문은 CTE를 정의하는 데 사용되며, 이 구문으로 작성된 CTE는 동일한 쿼리 내에서 여러 번 참조될 수 있습니다. 여러 CTE를 정의할 수도 있고, 이를 메인 쿼리와 결합하여 사용할 수도 있습니다.

```sql
WITH CTE_Sales AS (
    SELECT customer_id, SUM(total_amount) AS total_sales
    FROM orders
    GROUP BY customer_id
)
SELECT c.customer_name, s.total_sales
FROM customers c
INNER JOIN CTE_Sales s ON c.customer_id = s.customer_id;
```

설명: `CTE_Sales`라는 CTE를 정의하여 고객별 총 매출을 계산한 후,
메인 쿼리에서 이를 `customers` 테이블과 조인하여 고객 이름과 총 매출을 함께 반환합니다.

```sql
WITH RECURSIVE CTE_EmployeeHierarchy AS (
    SELECT employee_id, manager_id, employee_name, 1 AS level
    FROM employees
    WHERE manager_id IS NULL  -- 최상위 관리자를 찾음

    UNION ALL

    SELECT e.employee_id, e.manager_id, e.employee_name, eh.level + 1
    FROM employees e
    INNER JOIN CTE_EmployeeHierarchy eh ON e.manager_id = eh.employee_id
)
SELECT employee_id, employee_name, level
FROM CTE_EmployeeHierarchy;
```

설명: 재귀 CTE를 사용하여 직원들의 계층 구조를 나타냅니다. 최상위 관리자로부터 시작하여
각 직원의 계층을 계산하여 반환합니다.

```sql
WITH CTE_Orders AS (
    SELECT customer_id, COUNT(*) AS order_count
    FROM orders
    GROUP BY customer_id
),
CTE_Payments AS (
    SELECT customer_id, SUM(payment_amount) AS total_payments
    FROM payments
    GROUP BY customer_id
)
SELECT c.customer_name, o.order_count, p.total_payments
FROM customers c
LEFT JOIN CTE_Orders o ON c.customer_id = o.customer_id
LEFT JOIN CTE_Payments p ON c.customer_id = p.customer_id;
```

설명: 두 개의 CTE를 정의하여 하나는 고객별 주문 수를 계산하고,
다른 하나는 고객별 총 결제 금액을 계산한 후 이를 메인 쿼리에서 조합하여 반환합니다.

- **CASE WHEN, IF ELSE**

  데이터를 특정 조건에 따라 분류하거나, 새로운 컬럼을 계산해야 할 때 **CASE WHEN** 구문을 사용한다

```sql
SELECT customer_id,
  CASE
    WHEN COUNT(order_id)> 5 THEN 'VIP'
    ELSE 'Regular'
  END AS customer_type
FROM orders
GROUP BY customer_id;
```

- **RANK(), DENSE_RANK(), ROW_NUMBER()**
  - **RANK()**: 순위에 중복이 있으면 그 다음 순위가 건너뜀
  - **DENSE_RANK():** 순위에 중복이 있어도 그 다음 순위가 건너뛰지 않음
  - **ROW_NUMBER()**: 중복 없이 일련번호 매김

```sql
SELECT product_id,
  RANK() OVER(ORDER BY SUM(quantity) DESC) AS sales_rank
FROM orders
GROUP BY product_id;
```

- 날짜 처리함수

    - **TIMESTAMPDIFF()**: 두 날짜 또는 시간 간의 차이를 계산합니다.
    - **STR_TO_DATE()**: 문자열을 날짜 형식으로 변환할 때 사용합니다.
    - **DATE_FORMAT()**: 날짜를 특정 형식으로 출력합니다.

```
SELECT customer_id,
  TIMESTAMPDIFF(DAY,MIN(order_date),MAX(order_date)) AS days_between_orders
  FROM orders
  GROUP BY customer_id;
```

- **LEAD(), LAG()**

    - **LAG()** 함수: LAG() 함수는 현재 행의 이전 행에서 값을 가져옵니다. 주로 데이터의 변화를 비교하거나 시계열 데이터를 분석할 때 사용됩니다.
    - **LEAD()** 함수: LEAD() 함수는 현재 행의 다음 행에서 값을 가져옵니다. 주로 미래의 데이터를 참조하거나 다음 데이터를 비교할 때 사용됩니다.

```
SELECT order_id, customer_id, order_date,
       LAG(order_date, 1) OVER (PARTITION BY customer_id ORDER BY order_date)
       AS previous_order_date,
       LEAD(order_date, 1) OVER (PARTITION BY customer_id ORDER BY order_date)
       AS next_order_date
FROM orders;
```