[베이직반] SQL 3회차

구분	선택 학습반
날짜	@2025년 10월 2일 오후 7:00
대상	베이직
태그	SQL
튜터	강 강민정

[강의 자료]

베이직반_SQL_3회차_수업자료.pdf

[강의 녹화본]

📖 목차

1. 윈도우 함수 개념 및 사용 방법

🔥 모든 컬럼을 잃고 싶지 않을 때 사용합니다.

🗣️ 참고 자료: https://persistent-polyanthus-3e0.notion.site/SQL-5-24722d577d0e806694a8f10af9bfae5b?pvs=74

1-1. WINDOW 함수의 역할은?

💡
- 윈도우 함수는 행과 행의 관계를 알기 쉽게 해줍니다. (여러 행의 관계를 파악하기 위해 사용됩니다.)
- 분석함수 또는 순위 함수로 알려져있죠.

1️⃣ 정의

- 윈도우 함수는 행과 행 간의 관계를 정의하기 위해서 제공되는 함수

2️⃣ 역할

- 윈도우 함수를 사용해서 순위, 합계, 평균, 행 위치 등을 조작할 수 있습니다.

3️⃣ 특징

- 집계 함수는 `GROUP BY` 구문과 병행하여 사용할 수 있음.

4️⃣ 윈도우 함수 종류

분류	대표 함수	설명	GROUP BY 병행 여부
집계 함수	`SUM()`, `MAX()`, `MIN()`, `AVG()`, `COUNT()`	윈도우 범위 내 합계, 최대/최소, 평균, 개수 계산	가능
순위 함수	`RANK()`, `DENSE_RANK()`, `ROW_NUMBER()`	행에 순위를 매기거나 행 번호 부여	불가능
순서 함수	`FIRST_VALUE()`, `LAST_VALUE()`, `LAG()`, `LEAD()`	파티션 내 첫 값/마지막 값, 이전/다음 행 값 참조	불가능
비율/분석 함수	`RATIO_TO_REPORT()`, `PERCENT_RANK()`, `CUME_DIST()`, `NTILE(n)`	비율, 누적 백분율, 분위수 계산	불가능

▼ **GROUP BY 병행 여부에 대해 자세히 알아봅시다!**

- 구글 드라이브에서 emp.csv 다운로드 후, DBeaver에 데이터 불러오기(Data import) 해주세요.

[emp.csv 참고]

empno	ename	dept	salary
1001	KING	HR	5000
1002	BLAKE	SALES	3000
1003	CLARK	SALES	2800
1004	JONES	HR	4500
1005	SMITH	IT	4000
1006	ALLEN	SALES	3200

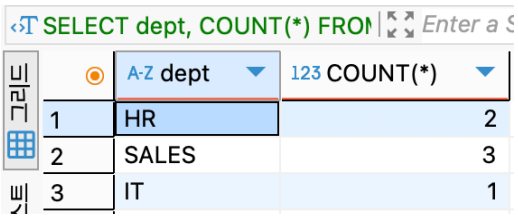
1. `GROUP BY`의 역할

- 여러 행을 특정 컬럼 값(또는 여러 컬럼 조합) 기준으로 묶어서 그룹 단위로 만듭니다.
  - `SUM`, `COUNT`, `AVG`, `MAX`, `MIN` 같은 **집계 함수**는 여러 행(집합)을 입력으로 받아 **단 하나의 값**을 출력
- 예를 들어, 부서별로 직원 수를 세면 **부서 단위로 행이 줄어듭니다.**

```
SELECT dept, COUNT(*)
FROM basic.emp
GROUP BY dept;
```

▼ 쿼리 실행 결과

- 부서별 1행씩만 남음. (즉, 원래 행이 사라지고 요약됨)



2. 윈도우 함수의 역할

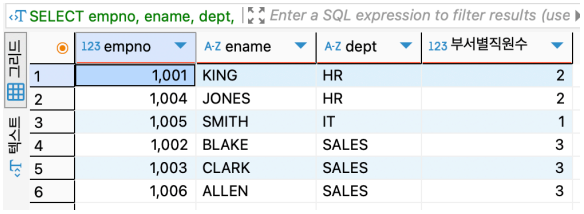
- 윈도우 함수는 **행을 그대로 유지하면서 추가적인 값을 보여주는 기능**입니다.
  - 예를 들어, 각 직원 옆에 본인의 "부서별 직원 수"를 같이 보여줄 수 있습니다.

```
SELECT empno, ename, dept,
```

```
    COUNT(*) OVER(PARTITION BY dept) AS 부서별직원수
FROM basic.emp;
```

▼ 쿼리 실행 결과

- 각 직원 행은 그대로 유지 + 부서별 직원 수 열이 생성됨.



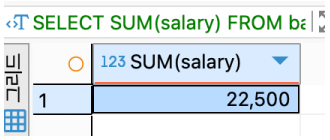
3. 왜 집계 함수만 `GROUP BY`와 병행 가능할까?

- `GROUP BY` 와 집계함수
  - `SUM` , `COUNT` , `AVG` , `MAX` , `MIN` 같은 집계 함수는 **여러 행을 묶어서 → 하나의 요약값**을 만드는 게 목적이에요.
  - 즉, 입력이 10행이든 100행이든, 결과는 **한 값**만 도출합니다.

```
SELECT SUM(salary) FROM basic.emp;
```

▼ 쿼리 실행 결과

- 전체 직원 월급을 다 더해서 **한 값**만 나옴

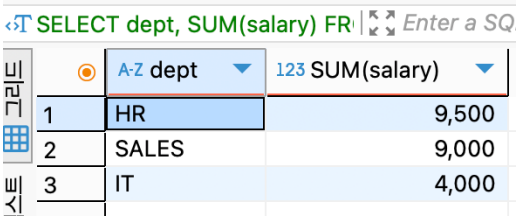


- GROUP BY의 역할
  - `GROUP BY` 는 특정 기준(예: 부서, 서버, 지역 등)으로 행들을 그룹(묶음)으로 나눕니다.
  - 각 그룹별로 집계 함수를 적용하면, **그룹 단위의 요약값**을 얻을 수 있습니다.

```
SELECT dept, SUM(salary)
FROM basic.emp
GROUP BY dept;
```

▼ 쿼리 실행 결과

- 부서별로 묶고 → 각 그룹마다 SUM 실행 → **부서별 총 월급**



- 반면, 윈도우 함수들 - 순위(RANK, ROW_NUMBER), 순서(LAG, LEAD), 비율 함수는 개별 행이 유지되어야 의미가 있습니다.

  → 그런데 `GROUP BY`는 행을 줄여버리죠.

  - 윈도우 함수는 각 행에 값을 붙이는 함수라서 행이 살아 있어야 합니다.
    - 순위 함수(`RANK`, `ROW_NUMBER`)는 **각 행마다 값을 붙이는 함수**예요.
    - 순서 함수(`LAG`, `LEAD`)도 **이전/다음 행을 참조해야 의미**가 있어요.
    - 비율 함수(`PERCENT_RANK`, `CUME_DIST`) 역시 **행 간 상대적 위치**가 필요합니다.

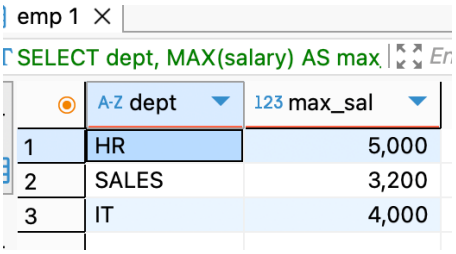
  하지만 `GROUP BY`는 행을 줄여버리기 때문에

  - **행이 사라진 상태**에서 순위를 매기거나, 이전/다음을 찾거나, 상대적 비율을 계산할 **대상이 없어집니다.**
  - 그래서 `GROUP BY`와 바로 병행이 불가능한 거예요.

구분	SQL 예시	결과
GROUP BY + 집계 함수	`SELECT dept, COUNT(*) FROM emp GROUP BY dept;`	부서별 1행만 남음
윈도우 함수 (행 유지)	`SELECT ename, dept, COUNT(*) OVER(PARTITION BY dept) FROM emp;`	직원 행 유지 + 부서별 직원 수 같이 표시
GROUP BY + 순위 함수	`SELECT dept, RANK() OVER(ORDER BY sal DESC) FROM emp GROUP BY dept;`	오류 발생 (GROUP BY는 행을 줄여버려서 순위 매길 대상이 없음)

```
-- 집계함수
SELECT
  dept,
  MAX(salary) AS max_sal
FROM basic.emp
GROUP BY dept;
```

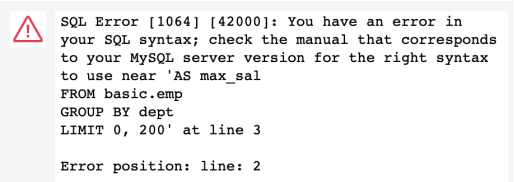
  - 쿼리 실행결과




```
-- 순위함수

SELECT
  dept,
  RANK() AS max_sal
FROM basic.emp
GROUP BY dept;
```

  - 쿼리 실행 결과



1-2. WINDOW 함수 문법은?

- **SELECT 절에서 사용!!**
  - 왜죠? **모든 행을 유지한 상태에서 계산 결과를 컬럼처럼 붙이는 것**이 기본 목적입니다!
- 윈도우 함수 기본 쿼리 작성법

```
-- 윈도우 함수 기본 문법

SELECT
   WINDOW_FUNCTION(컬럼명) OVER ( -- OVER는 윈도우 함수가 계산될 범위를 정하는 키워드
      PARTITION BY 컬럼명   -- 파티션 기준 그룹화
      ORDER BY 컬럼명       -- 파티션 내 정렬 기준
   ) AS 별칭
FROM
   테이블명;


SELECT WINDOW_FUNCTION(컬럼명) OVER (PARTITION BY 컬럼명 ORDER BY 컬럼명) AS 별칭
FROM 테이블명;

예시) ROW_NUMBER() OVER(PARTITION BY 컬럼1 ORDER BY 컬럼2)
→ ROW_NUMBER 라는 윈도우 함수를 사용하고, 그 기준을 컬럼1로 , 정렬은 컬럼2로 지정
```

구성요소	설명
`OVER`	윈도우 함수 범위를 지정하는 핵심 키워드
`PARTITION BY`	그룹핑 기준 (마치 `GROUP BY`처럼 작동하지만, 행은 그대로 유지)
`ORDER BY`	정렬 기준 (순위, 누적 계산 등에 필요)

▼ **윈도우 함수 괄호 안에 컬럼 필요 여부 정리** 🧯

- 괄호에 `()`만 쓰는 함수는 주로 **순위를 부여**하는 함수들이에요.
  - 보통 순위만 매기면 되니까, 특정 컬럼 계산을 목적으로 하지 않음.
- 괄호 안에 `컬럼`이 들어가는 함수는 **특정 값을 기준으로 비교, 계산**하는 함수입니다.

함수 이름	괄호 안에 컬럼 필요 여부	설명	쿼리 예시
RANK()	❌ 없음	순위 (중복 순위 있음, 건너뜀)	`RANK() OVER (ORDER BY salary DESC) AS rank_no`
DENSE_RANK()	❌ 없음	순위 (중복 순위 있음, 건너뜀 없음)	`DENSE_RANK() OVER (ORDER BY salary DESC) AS dense_rank_no`
ROW_NUMBER()	❌ 없음	고유 순번 부여	`ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_no`
PERCENT_RANK()	❌ 없음	백분율 순위 (0~1 사이)	`PERCENT_RANK() OVER (ORDER BY salary DESC) AS pct_rank`
CUME_DIST()	❌ 없음	누적 백분율 (현재 행 이하 비율)	`CUME_DIST() OVER (ORDER BY salary DESC) AS cume_dist`
NTILE(N)	✅ 있음 (N은 숫자)	N등분으로 나눠 등급 부여	`NTILE(4) OVER (ORDER BY salary DESC) AS quartile`
LAG(컬럼)	✅ 있음	이전 행의 값	`LAG(salary) OVER (ORDER BY salary) AS prev_salary`
LAG(컬럼, 숫자)	✅ 있음	N번째 이전 행의 값	`LAG(salary, 2) OVER (ORDER BY salary) AS prev2_salary`
LEAD(컬럼)	✅ 있음	이후 행의 값	`LEAD(salary) OVER (ORDER BY salary) AS next_salary`
LEAD(컬럼, 숫자)	✅ 있음	N번째 이후 행의 값	`LEAD(salary, 3) OVER (ORDER BY salary) AS next3_salary`
FIRST_VALUE(컬럼)	✅ 있음	파티션 내 첫 번째 값	`FIRST_VALUE(salary) OVER (PARTITION BY department_id ORDER BY salary DESC) AS top_salary`

함수 이름	괄호 안에 컬럼 필요 여부	설명	쿼리 예시
LAST_VALUE(컬럼)	✅ 있음	파티션 내 마지막 값	`LAST_VALUE(salary) OVER (PARTITION BY department_id ORDER BY salary DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS bottom_salary`
RATIO_TO_REPORT(컬럼)	✅ 있음	전체합 대비 현재값 비율 (Oracle)	`RATIO_TO_REPORT(salary) OVER () AS salary_ratio`
SUM(컬럼)	✅ 있음	누적 합계 (윈도우 함수일 경우)	`SUM(salary) OVER (PARTITION BY department_id ORDER BY salary) AS running_total`
AVG(컬럼)	✅ 있음	누적 평균 등	`AVG(salary) OVER (PARTITION BY department_id ORDER BY salary) AS running_avg`

- 윈도우 함수(Window Function) 기본 쿼리 예시

```
SELECT
   컬럼명,
   윈도우함수(컬럼명) OVER (
      PARTITION BY 컬럼명   -- 선택사항
      ORDER BY 컬럼명       -- 정렬 기준
   ) AS 별칭
FROM
   테이블명
WHERE
   조건식
ORDER BY
   컬럼명 DESC,
   컬럼명 ASC;
```

```
SELECT
   employee_id,
   department_id,
   salary,
   RANK() OVER (
      PARTITION BY department_id
      ORDER BY salary DESC
   ) AS salary_rank
FROM
   employees
ORDER BY
   department_id ASC,
   salary DESC;
```

- `employees` 테이블에 포함된 컬럼
  - employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary, commission_pct, manager_id, department_id

예시 문제) 부서(department_id)별 직원 급여 순위를 구하라!

▼ 쿼리 해석

- **SELECT employee_id, department_id, salary**

  → 직원 ID, 부서 ID, 급여 컬럼을 조회

- **RANK() OVER (...) AS salary_rank**
  - `RANK()` 함수는 순위를 매기는 윈도우 함수입니다.
  - `PARTITION BY department_id` → 같은 부서(department_id)별로 순위를 나눕니다. 즉, 부서별 랭킹!
  - `ORDER BY salary DESC` → 급여(salary)를 내림차순(높은 값 먼저)으로 정렬한 순서대로 순위를 매깁니다.
  - 결과적으로 **각 부서마다 급여가 높은 직원이 1위, 그 다음이 2위 ...** 로 순위가 매겨집니다.
  - 단, **동일 급여일 경우 동일 순위를 부여**하고, 그 다음 순위는 건너뜁니다.

    예시) 5000(1위), 4000(2위), 4000(2위), 3000(4위)

- **FROM employees**

  → 데이터가 있는 테이블은 `employees`

- **ORDER BY department_id ASC, salary DESC**

- 최종 결과를 출력할 때 **부서 ID 기준 오름차순 정렬**
- 같은 부서 안에서는 **급여 내림차순**으로 정렬
- 즉, 각 부서별로 높은 급여 직원이 먼저 나오도록 정렬

▼ 쿼리 실행 결과 예시

employee_id	department_id	salary	salary_rank
101	10	5000	1
102	10	4000	2
103	10	4000	2
104	10	3000	4
201	20	6000	1
202	20	5500	2
203	20	4000	3

- 집계 함수(Group Function), 윈도우 함수(Window Function), GROUP BY, HAVING 등을 모두 사용하는 복합 쿼리 기본 구조 예시

```
/* 집계 함수(Group Function),
윈도우 함수(Window Function),
GROUP BY, HAVING 등을 모두 사용하는 복합
쿼리 기본 구조 예시 */

SELECT
   컬럼명,
   그룹함수(컬럼명),
   윈도우함수 OVER (
      PARTITION BY 컬럼명
      ORDER BY 컬럼명
   ) AS 별칭
FROM
   테이블명
WHERE
   조건식
GROUP BY
   컬럼명
HAVING
   조건식   -- GROUP BY에 대한 조건
ORDER BY
   컬럼명 DESC,
   컬럼명 ASC;
```

```
SELECT
   department_id,
   SUM(salary) AS dept_total_salary,
   RANK() OVER (
      PARTITION BY department_id
      ORDER BY salary DESC
   ) AS salary_rank
FROM
   employees
WHERE
   salary > 3000
GROUP BY
   department_id, salary
HAVING
   SUM(salary) > 10000
ORDER BY
   department_id DESC,
   salary ASC;
```

- `employees` 테이블에 포함된 컬럼
  - employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary, commission_pct, manager_id, department_id

예시 문제) 부서별 합계 + 개인 급여 순위 구하라!

▼ 쿼리 해석

- **FROM employees**
  - 데이터 소스는 `employees` 테이블(직원 정보 테이블)
- **WHERE salary > 3000**
  - 급여가 3000 초과인 직원만 대상
- **GROUP BY department_id, salary**
  - 데이터를 **부서별(** `department_id` **) + 급여별(** `salary` **)** 조합으로 그룹화 진행

    - 즉, 같은 부서 내에서 같은 급여를 받는 직원들이 하나의 그룹이 됨.
- **SELECT SUM(salary) AS dept_total_salary**
    - 그룹별로 `salary`를 합산!
    - 예를 들어 같은 부서에 급여 4000을 받는 직원이 2명 있다면, `SUM(salary)`는 8000이 됨.
- **HAVING SUM(salary) > 10000**
    - 그룹핑된 결과 중에서 급여 합계가 10,000을 넘는 그룹만 남김.
    - 즉, 부서별-급여 조합의 총액이 10,000 이하라면 제외
- **RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_rank**
    - 윈도우 함수 `RANK()`를 이용해, **같은 부서 내에서 급여 순위를 매깁니다.**
    - `PARTITION BY department_id` → 부서별로 순위 계산
    - `ORDER BY salary DESC` → 급여가 높은 순으로 순위 결정
    - 동점자는 같은 순위를 받고, 다음 순위는 건너뜀.
- **ORDER BY department_id DESC, salary ASC**
    - 최종 출력 시 부서번호는 내림차순, 같은 부서 안에서는 급여를 오름차순으로 정렬

▼ 쿼리 실행 결과 예시

department_id	salary	dept_total_salary	salary_rank
20	6000	12000	1
20	5000	15000	2
10	4000	12000	1
10	3500	10500	2

1-3. 윈도우 함수 별 정의 & 특징 요약

- ⭐는 가장 많이 쓰는 함수입니다!

함수	정의	특징
RANK()	`ORDER BY` 기준 순위 부여	동순위 → 같은 값, 다음 순위 건너뜀 (예: 1,2,2,4)
DENSE_RANK()	`RANK`와 동일하나 순위 건너뛰지 않음	동순위 → 같은 값, 다음 순위 연속 (예: 1,2,2,3)
ROW_NUMBER() ⭐	각 행에 고유 번호 부여	중복값 있어도 고유 번호 (예: 1,2,3,4)
FIRST_VALUE()	파티션(그룹) 내에서 정렬 기준상 첫 번째 값을 반환	파티션별 정렬 후 첫 번째 값 가져옴
LAST_VALUE()	파티션(그룹) 내에서 정렬 기준상 마지막 값을 반환	기본은 현재 행까지의 마지막 값 → 옵션 필요
LAG() ⭐	이전 행 값 가져옴	기본 `N=1`, 없으면 NULL
LEAD() ⭐	다음 행 값 가져옴	기본 `N=1`, 없으면 NULL
PERCENT_RANK()	백분위 순위(0~1)	(순위-1) / (전체행수-1)
CUME_DIST()	누적 비율(0~1)	현재 값 이하 비율 → 항상 마지막 값은 1
NTILE(N)	데이터를 N등분해 구간 번호 부여	1~N 등분, 분위수 계산에 활용
RATIO_TO_REPORT()	전체합 대비 비율	Oracle 전용, 합계 1.0 (100%)

▼ FIRST_VALUE() 와 LAST_VALUE() 다시 점검하기

1. `employees` 테이블

department_id	employee_name	salary
10	Ahn	3000

department_id	employee_name	salary
10	Kim	5000
10	Lee	4000
20	Park	4500
20	Choi	7000
20	Jung	6000

2. FIRST_VALUE()

- 부서별(`department_id`)로 묶고, 급여(`salary`) 높은 순으로 정렬했을 때 **첫 번째 값(= 최고 급여자)** 을 가져오기

```
SELECT
    department_id,
    employee_name,
    salary,
    FIRST_VALUE(employee_name) OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
    ) AS top_earner
FROM employees;
```

- 쿼리 실행 결과 예시

department_id	employee_name	salary	top_earner
10	Kim	5000	**Kim**
10	Lee	4000	**Kim**
10	Ahn	3000	**Kim**
20	Choi	7000	**Choi**
20	Jung	6000	**Choi**
20	Park	4500	**Choi**

`FIRST_VALUE()` 는 그룹별로 정렬했을 때 **맨 처음 값(최고 급여자)** 를 모든 행에 붙여줍니다.

3. LAST_VALUE()

- 같은 방식으로, 부서별(`department_id`) 그룹에서 급여 내림차순 정렬 후 **마지막 값(= 최저 급여자)** 을 가져오기

```
SELECT
    department_id,
    employee_name,
    salary,
    LAST_VALUE(employee_name) OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING -- 중요!
    ) AS lowest_earner
FROM employees;
```

- 쿼리 실행 결과 예시

department_id	employee_name	salary	lowest_earner
10	Kim	5000	**Ahn**
10	Lee	4000	**Ahn**
10	Ahn	3000	**Ahn**

department_id	employee_name	salary	lowest_earner
20	Choi	7000	**Park**
20	Jung	6000	**Park**
20	Park	4500	**Park**

`LAST_VALUE()` 는 그룹 내 정렬 결과의 **마지막 값(최저 급여자)** 을 반환합니다.

그냥 `LAST_VALUE()` 만 쓰면 "현재 행까지의 마지막 값"만 나오므로 원하는 값이 안 나올 수 있어요.
그래서 보통 `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` 옵션을 함께 써서 파티션 전체 범위 를 지정해야 합니다. 잊지 말아주세요!!

1-4. 윈도우 함수를 쓰는 대표적인 상황들

1. **순위 매기기 (Ranking)**

- 문제: 부서별 급여 순위를 매겨라
- 해결: `RANK()` , `DENSE_RANK()` , `ROW_NUMBER()` 로 해결해볼 수 있어요.

```
SELECT department_id, employee_name, salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_rank
FROM employees;
```

→ 부서별로 높은 급여 순위를 계산하면서, **모든 직원 행을 그대로 유지**

2. **누적값, 이동평균 (Running / Moving Aggregation)**

- 문제: 월별 매출 누적 합계를 구하라
- 해결: `SUM() OVER` , `AVG() OVER` 로 해결해볼 수 있어요.

```
SELECT month, sales,
    SUM(sales) OVER (ORDER BY month) AS running_total
FROM sales_data;
```

→ 각 달 매출 행은 그대로 두고, 누적 합계 컬럼을 추가

3. **비교 (이전/다음 행 값 참조)**

- 문제: 직원의 급여가 직전 직원보다 얼마나 증가했는가?
- 해결: `LAG()` , `LEAD()` 로 해결해볼 수 있어요.

```
SELECT employee_name, salary,
    LAG(salary) OVER (ORDER BY hire_date) AS prev_salary,
    salary - LAG(salary) OVER (ORDER BY hire_date) AS diff
FROM employees;
```

→ 같은 테이블 내 행 간 비교 가능 (SQL에서 자기 자신과 바로 위 행 비교가 가능해짐)

4. **비율 계산**

- 문제: 부서별 급여에서 각 직원 급여가 차지하는 비율
- 해결: `RATIO_TO_REPORT()`, `PERCENT_RANK()`, `CUME_DIST()` 로 해결해볼 수 있어요.

```
SELECT department_id,
    employee_name,
    salary,
    salary / SUM(salary) OVER (PARTITION BY department_id) AS salary_ratio
FROM employees;
```

→ 각 행의 급여가 부서 전체 합계에서 몇 %인지 계산

5. **특정 위치 값 가져오기**

- 문제: 부서별 최고 급여자, 최저 급여자 표시
- 해결: `FIRST_VALUE()`, `LAST_VALUE()` 로 해결해볼 수 있어요.

```
SELECT department_id, employee_name, salary,
    FIRST_VALUE(employee_name) OVER (PARTITION BY department_id ORDER BY salary DESC) AS top_earner,
    LAST_VALUE(employee_name) OVER (PARTITION BY department_id ORDER BY salary DESC ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lowest_earner
FROM employees;
```

→ 파티션(그룹) 내에서 정렬 기준으로 **맨 앞/맨 뒤 행의 값**을 가져올 수 있음.

🍀 윈도우 함수는 **데이터를 줄이지 않고 행 그대로 유지하면서 추가적인 통계·순위·누적값·비교값을 컬럼으로 붙이고 싶을 때** 씁니다.

- 집계 함수(`SUM`, `AVG`)는 데이터를 **줄이고 요약**
- 윈도우 함수는 데이터를 **그대로 두고 확장**

2. 문제 풀이

▼ 문제 1번 (코드카타 기준 48번)

🗣️ 문제 링크: https://school.programmers.co.kr/learn/courses/30/lessons/59044

다음은 식당의 정보를 담은 `REST_INFO` 테이블입니다.
`REST_INFO` 테이블은 다음과 같으며 `REST_ID`, `REST_NAME`, `FOOD_TYPE`, `VIEWS`, `FAVORITES`, `PARKING_LOT`, `ADDRESS`, `TEL`은 식당 ID, 식당 이름, 음식 종류, 조회수, 즐겨찾기수, 주차장 유무, 주소, 전화번호를 의미합니다.

Column name	Type	Nullable
REST_ID	VARCHAR(5)	FALSE
REST_NAME	VARCHAR(50)	FALSE
FOOD_TYPE	VARCHAR(20)	TRUE
VIEWS	NUMBER	TRUE
FAVORITES	NUMBER	TRUE
PARKING_LOT	VARCHAR(1)	TRUE
ADDRESS	VARCHAR(100)	TRUE
TEL	VARCHAR(100)	TRUE

- 문제

`REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.

- 예시

`REST_INFO` 테이블이 다음과 같을 때

REST_ID	REST_NAME	FOOD_TYPE	VIEWS	FAVORITES	PARKING_LOT	ADDRESS
00001	은돼지식당	한식	1150345	734	N	서울특별시 중구 다산로 149
00002	하이가쯔네	일식	120034	112	N	서울시 중구 신당동 375-21
00003	따띠따띠뜨	양식	1234023	102	N	서울시 강남구 신사동 627-3 1F
00004	스시사카우스	일식	1522074	230	N	서울시 서울시 강남구 신사동 627 27
00005	코슢스	일식	15301	123	N	서울특별시 강남구 언주로153길

SQL을 실행하면 다음과 같이 출력되어야 합니다.

FOOD_TYPE	REST_ID	REST_NAME	FAVORITES
한식	00001	은돼지식당	734
일식	00004	스시사카우스	230
양식	00003	따띠따띠뜨	102

▼ 문제 풀이 아이디어

❓ **음식 종류별로 즐겨찾기 수가 가장 많은 식당을 찾고**, 그 결과를 음식 종류 기준 내림차순으로 정렬하는 SQL을 작성하자!

- `REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.
  - 음식 종류(FOOD_TYPE)별로 즐겨찾기 수(FAVORITES)가 가장 많은 식당을 찾는다.

- FOOD_TYPE, REST_ID, REST_NAME, FAVORITES
- 음식 종류를 기준으로 내림차순 정렬

→ 그룹별 최대값을 뽑아내는 문제입니다.

▼ 정답 쿼리

🤓 **FROM → ON → JOIN → WHERE → GROUP BY → 집계함수 → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT**

✅ **1. WHERE 절 서브쿼리 방식 = 중첩(일반) 서브쿼리**

- 정답 쿼리

```
SELECT FOOD_TYPE, REST_ID, REST_NAME, FAVORITES
FROM REST_INFO R
WHERE FAVORITES = (
   SELECT MAX(FAVORITES)
   FROM REST_INFO
   WHERE FOOD_TYPE = R.FOOD_TYPE
)
ORDER BY FOOD_TYPE DESC;
```

▼ 쿼리 해석

`REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL 문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.

1. 바깥 쿼리

```
SELECT FOOD_TYPE, REST_ID, REST_NAME, FAVORITES  -- 최종적으로 가져와야하는 컬럼들
FROM REST_INFO R
```

2. `WHERE` 절 서브쿼리

```
WHERE FAVORITES = (
   SELECT MAX(FAVORITES)
   FROM REST_INFO
   WHERE FOOD_TYPE = R.FOOD_TYPE
)
```

▼ 🤔 **왜 이 문제에서 WHERE 절 서브쿼리를 쓸 수 있을까요?**

👋

- 서브쿼리 결과에 따라 바깥쪽 `WHERE` 조건이 결정되는 방식
  1. 서브쿼리가 먼저 실행되어 **값을 반환하고,**
  2. 메인쿼리는 **그 반환된 값**을 기준으로 `WHERE` 조건을 적용한다는 뜻!

- 문제에서 WHERE 절 서브쿼리를 써보면 좋겠다는 힌트는?
  - **즐겨찾기 수가 가장 많은**

- 우리가 최종적으로 가져와야 하는 컬럼은 `FOOD_TYPE`, `REST_ID`, `REST_NAME`, `FAVORITES`
- 이 중 각 `FOOD_TYPE` 그룹마다 최댓값 조건이 적용되는 건 `FAVORITES` 컬럼 밖에 없음.
- `FOOD_TYPE`에 여러 행이 존재하므로, 단순히 최댓값 숫자만 지정해두면 전체적으로 오답이 됨.
- 따라서 각 `FOOD_TYPE` 그룹에서 `FAVORITES`의 최댓값을 만족하는 행을 걸러내야 함.
- 이런 경우에 사용할 수 있는 방법이 바로 WHERE 절 서브쿼리

- 바깥 쿼리(`R`)의 각 행에 대해, 같은 음식 종류(`R.FOOD_TYPE`) 에서 **최대 FAVORITES** 값을 찾음.
- 즉, **음식 종류별로 즐겨찾기 수 최댓값을 가진 행**만 필터링됨.

3. 정렬

```
ORDER BY FOOD_TYPE DESC;
```

- 최종적으로 음식 종류(`FOOD_TYPE`)를 기준으로 **내림차순 정렬**

✅ **2. 서브쿼리로 음식 종류별 즐겨찾기 최대값을 구한 뒤, 원본 테이블과 조인해서 해당 식당의 정보를 가져오는 방식**

- 정답 쿼리

```
SELECT R.FOOD_TYPE, R.REST_ID, R.REST_NAME, R.FAVORITES
FROM REST_INFO R
JOIN (
   SELECT FOOD_TYPE, MAX(FAVORITES) AS MAX_FAV
   FROM REST_INFO
   GROUP BY FOOD_TYPE
) M
ON R.FOOD_TYPE = M.FOOD_TYPE
AND R.FAVORITES = M.MAX_FAV
ORDER BY R.FOOD_TYPE DESC;
```

▼ 쿼리 해석

`REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL 문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.

1. 내부 서브쿼리 (`M`)

```
SELECT FOOD_TYPE, MAX(FAVORITES) AS MAX_FAV
FROM REST_INFO
GROUP BY FOOD_TYPE
```

- `REST_INFO` 테이블에서 음식 종류별(FOOD_TYPE)로 그룹핑
- 각 그룹에서 `FAVORITES`(즐겨찾기 수)의 최댓값(`MAX(FAVORITES)`)을 구한다.
- 결과는? 음식 종류별로 즐겨찾기 수가 가장 큰 값만 남긴다.
- 결과)

FOOD_TYPE	MAX_FAV
한식	734
일식	230

FOOD_TYPE	MAX_FAV
양식	102

2. JOIN (`R JOIN M`)

```
FROM REST_INFO R
JOIN (
   SELECT FOOD_TYPE, MAX(FAVORITES) AS MA
X_FAV
   FROM REST_INFO
   GROUP BY FOOD_TYPE
) M
ON R.FOOD_TYPE = M.FOOD_TYPE
AND R.FAVORITES = M.MAX_FAV
```

```
-- INNER JOIN

SELECT
a.컬럼1,
b.컬럼2 ...
FROM 테이블명1 AS a
JOIN 테이블명2 AS b
ON a.공통컬럼 = b.공통컬럼;
```

- 원본 테이블 `REST_INFO` 를 `R` 이라는 별칭으로 사용
- 서브쿼리의 결과인 `M` 과 조인한다. (여기서 서브쿼리 결과인 `M` 은 임시테이블)
- 조인 조건 (INNER JOIN)
  - 음식 종류가 같고 (`R.FOOD_TYPE = M.FOOD_TYPE`)
  - 해당 음식 종류의 **즐겨찾기 수가 최대값**인 경우만 매칭 (`R.FAVORITES = M.MAX_FAV`)
- 결과는? 각 음식 종류에서 즐겨찾기 수가 가장 많은 식당만 필터링된다.

FOOD_TYPE	REST_ID	REST_NAME	FAVORITES
한식	00001	은돼지식당	734
일식	00004	스시사카우스	230
양식	00003	따띠따띠뜨	102

3. 최종 SELECT & ORDER

```
SELECT R.FOOD_TYPE, R.REST_ID, R.REST_NAME, R.FAVORITES
ORDER BY R.FOOD_TYPE DESC
```

- 필요한 컬럼만 선택해서 출력: `R.FOOD_TYPE` , `R.REST_ID` , `R.REST_NAME` , `R.FAVORITES`
- 음식 종류(`FOOD_TYPE`)를 **내림차순**으로 정렬
- 쿼리 최종 실행 결과

FOOD_TYPE	REST_ID	REST_NAME	FAVORITES
한식	00001	은돼지식당	734
일식	00004	스시사카우스	230
양식	00003	따띠따띠뜨	102

✅ **3. CTE + JOIN 방식**

- 정답 쿼리

```
WITH MAX_FAV AS (
   SELECT FOOD_TYPE, MAX(FAVORITES) AS MAX_FAV
   FROM REST_INFO
```

```
    GROUP BY FOOD_TYPE
)
SELECT R.FOOD_TYPE, R.REST_ID, R.REST_NAME, R.FAVORITES
FROM REST_INFO R
JOIN MAX_FAV M
  ON R.FOOD_TYPE = M.FOOD_TYPE
 AND R.FAVORITES = M.MAX_FAV
ORDER BY R.FOOD_TYPE DESC;
```

▼ 쿼리 해석

`REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.

1. CTE

```
WITH MAX_FAV AS (
    SELECT FOOD_TYPE, MAX(FAVORITES) AS MAX_FAV
    FROM REST_INFO
    GROUP BY FOOD_TYPE
)
```

- `MAX_FAV` 라는 이름의 임시 테이블(CTE)을 생성
- 여기에는 음식 종류별로 즐겨찾기 수의 최댓값만 남습니다!
- 실행 결과)

FOOD_TYPE	MAX_FAV
한식	734
일식	230
양식	102

2. 메인 쿼리

```
SELECT R.FOOD_TYPE, R.REST_ID, R.REST_NAME, R.FAVORITES
FROM REST_INFO R
JOIN MAX_FAV M
  ON R.FOOD_TYPE = M.FOOD_TYPE
 AND R.FAVORITES = M.MAX_FAV
ORDER BY R.FOOD_TYPE DESC;
```

- 원본 테이블(`REST_INFO`)을 `R` 이라 부르고, CTE `MAX_FAV` 와 `INNER JOIN` 을 합니다.
- 조인 조건
  - 음식 종류가 같고 (`R.FOOD_TYPE = M.FOOD_TYPE`)
  - 즐겨찾기 수가 그 음식 종류의 최댓값과 같을 때 (`R.FAVORITES = M.MAX_FAV`)

  → 이렇게 하면, 음식 종류별로 **즐겨찾기 수가 가장 많은 행**만 남습니다.

✅ 4. 윈도우 함수 방식

- 정답 쿼리

```
SELECT FOOD_TYPE, REST_ID, REST_NAME, FAVORITES
FROM (
```

```
    SELECT
        FOOD_TYPE,
        REST_ID,
        REST_NAME,
        FAVORITES,
        ROW_NUMBER() OVER (PARTITION BY FOOD_TYPE ORDER BY FAVORITES DESC) AS RN
    FROM REST_INFO
) A
WHERE RN = 1
ORDER BY FOOD_TYPE DESC;
```

▼ 쿼리 해석

`REST_INFO` 테이블에서 **음식종류 별**로 **즐겨찾기 수가 가장 많은** 식당의 **음식 종류, ID, 식당 이름, 즐겨찾기 수를 조회**하는 SQL문을 작성해주세요. 이때 결과는 **음식 종류를 기준으로 내림차순 정렬**해주세요.

▼ 🤔 **왜 이 문제에서 윈도우 함수를 쓸 수 있을까요?**

1. 문제 요구사항

- 음식 종류별(FOOD_TYPE) 그룹 안에서 즐겨찾기 수(FAVORITES)가 가장 큰 행을 찾겠다.
- 즉, **그룹 내에서 순위를 매기고 1등만 뽑는 문제**예요.

2. 윈도우 함수의 특징

- `OVER (PARTITION BY FOOD_TYPE ORDER BY FAVORITES DESC)` 구문을 쓰면
  - 그룹(FOOD_TYPE)별로 행을 나눠서 정렬 기준(FAVORITES DESC)에 따라 순서를 매길 수 있음.

3. 이 문제에 활용할 수 있는 이유

- 우리는 그룹별 최댓값을 구하고 싶음 → 즉, 각 그룹에서 **1등만 뽑으면 됩니다.**
- `ROW_NUMBER() OVER (PARTITION BY FOOD_TYPE ORDER BY FAVORITES DESC) AS RN`
  → 그룹별로 FAVORITES 내림차순 정렬
- `WHERE RN=1` 하면 각 그룹의 최대 즐겨찾기 수를 가진 행만 선택 가능.

함수	정의	특징
RANK()	`ORDER BY` 기준 순위 부여	동순위 → 같은 값, 다음 순위 건너뜀 (예: 1,2,2,4)
DENSE_RANK()	`RANK`와 동일하나 순위 건너뛰지 않음	동순위 → 같은 값, 다음 순위 연속 (예: 1,2,2,3)
ROW_NUMBER()	각 행에 고유 번호 부여	중복값 있어도 고유 번호 (예: 1,2,3,4)

1. **FROM 절 서브쿼리 = 인라인 뷰**

```
SELECT
    FOOD_TYPE,
    REST_ID,
    REST_NAME,
    FAVORITES,
    ROW_NUMBER() OVER (PARTITION BY FOOD_TYPE ORDER BY FAVORITES DESC) AS RN
FROM REST_INFO
```

- `ROW_NUMBER() OVER (...)` : 윈도우 함수 → 각 행에 순번을 매김

- `PARTITION BY FOOD_TYPE` : 음식 종류별로 그룹을 나눔
- `ORDER BY FAVORITES DESC` : 그룹 안에서 즐겨찾기 수가 큰 순서대로 번호를 매김
- 결과적으로 각 음식 종류 그룹에서 **즐겨찾기 수 1등인 행이 RN = 1**
- **참고) 왜 `RANK()` 나 `DENSE_RANK()` 가 아닌가?** 🤔
  - `RANK()` → 동점이 있으면 같은 순위(1, 1, 3...)
  - `DENSE_RANK()` → 동점이 있으면 같은 순위지만 다음 순위는 바로 이어짐(1, 1, 2...)
  - `ROW_NUMBER()` → **무조건 1개만 뽑힘** (동점도 강제로 순서가 매겨짐 → 1, 2, 3...)
- **결과 예시)**

FOOD_TYPE	REST_ID	REST_NAME	FAVORITES	RN
일식	00004	스시사카우스	230	1
일식	00005	코슌스	123	2
일식	00002	하이가쯔네	112	3

2. 외부 쿼리

```
SELECT FOOD_TYPE, REST_ID, REST_NAME, FAVORITES
FROM (...) A
WHERE RN = 1
```

- 내부 쿼리 결과에서 `RN = 1` 인 행만 선택
- 즉, 각 음식 종류 그룹에서 **즐겨찾기 수가 가장 큰 식당만 남김**

3. 정렬

```
ORDER BY FOOD_TYPE DESC;
```

- 최종 결과를 음식 종류 기준으로 내림차순 정렬

4. 최종 실행 결과

FOOD_TYPE	REST_ID	REST_NAME	FAVORITES
한식	00001	은돼지식당	734
일식	00004	스시사카우스	230
양식	00003	따띠따띠뜨	102

- **WHERE 절 서브쿼리 방식 / JOIN 방식 / 윈도우 함수 방식**

구분	WHERE 절 서브쿼리 방식	JOIN 방식	윈도우 함수 방식
동작 원리	바깥 테이블의 각 행마다 서브쿼리를 실행해 해당 FOOD_TYPE의 `MAX(FAVORITES)` 와 비교	서브쿼리에서 음식 종류별 최댓값 테이블(M)을 만들고, 원본 테이블과 매칭	`ROW_NUMBER() OVER (PARTITION BY FOOD_TYPE ORDER BY FAVORITES DESC)` 로 그룹별 순위를 매기고, 1등만 선택
가독성	간단하고 직관적 (짧은 코드)	최댓값 테이블 → 조인 구조가 명확	"그룹별 순위 매기고 1등 뽑는다"는 개념이 직관적
성능	바깥 행마다 서브쿼리 실행 → 대용량에서 비효율적	그룹별 최댓값을 한 번만 계산 후 조인 → 효율적	순위 계산을 한 번만 하고 바로 필터링 → 대규모 데이터에도 효율적
확장성	단순 최대값 비교까지만 적합	평균, 최소값 등 다른 집계값을 함께 활용하기 쉽다	다양한 순위 로직(RANK, DENSE_RANK) 적용 가능 → 동점 처리 유연
동점 처리	최대값과 같은 여러 행 모두 출력됨	최대값과 같은 여러 행 모두 출력됨	`ROW_NUMBER()` 는 동점 중 1개만, `RANK` / `DENSE_RANK` 는 동점 모두 출력 가

구분	WHERE 절 서브쿼리 방식	JOIN 방식	윈도우 함수 방식
			능
권장 상황	데이터가 적거나 학습용으로 간단히 쓸 때	실무에서 집계값을 다른 조건과 함께 활용할 때	실무에서 직관적이고 확장성·성능까지 모두 고려할 때

▼ 문제 2번 (코드카타 기준 49번)

🗣️ 문제 링크: https://school.programmers.co.kr/learn/courses/30/lessons/59044

다음은 식품의 정보를 담은 `FOOD_PRODUCT` 테이블입니다.
`FOOD_PRODUCT` 테이블은 다음과 같으며 `PRODUCT_ID`, `PRODUCT_NAME`, `PRODUCT_CD`, `CATEGORY`, `PRICE`는 식품 ID, 식품 이름, 식품코드, 식품분류, 식품 가격을 의미합니다.

Column name	Type	Nullable
PRODUCT_ID	VARCHAR(10)	FALSE
PRODUCT_NAME	VARCHAR(50)	FALSE
PRODUCT_CD	VARCHAR(10)	TRUE
CATEGORY	VARCHAR(10)	TRUE
PRICE	NUMBER	TRUE

- 문제

`FOOD_PRODUCT` 테이블에서 **식품분류 별**로 **가격이 제일 비싼 식품의 분류, 가격, 이름을 조회**하는 SQL문을 작성해주세요. 이때 **식품분류가 '과자', '국', '김치', '식용유'인 경우만 출력**시켜 주시고 결과는 **식품 가격을 기준으로 내림차순 정렬**해주세요.

- 예시

`FOOD_PRODUCT` 테이블이 다음과 같을 때

PRODUCT_ID	PRODUCT_NAME	PRODUCT_CD	CATEGORY	PRICE
P0018	맛있는고추기름	CD_OL00008	식용유	6100
P0019	맛있는카놀라유	CD_OL00009	식용유	5100
P0020	맛있는산초유	CD_OL00010	식용유	6500
P0021	맛있는케첩	CD_SC00001	소스	4500
P0022	맛있는마요네즈	CD_SC00002	소스	4700
P0039	맛있는황도	CD_CN00008	캔	4100
P0040	맛있는명이나물	CD_CN00009	캔	3500
P0041	맛있는보리차	CD_TE00010	차	3400
P0042	맛있는메밀차	CD_TE00001	차	3500
P0099	맛있는맛동산	CD_CK00002	과자	1800

SQL을 실행하면 다음과 같이 출력되어야 합니다.

CATEGORY	MAX_PRICE	PRODUCT_NAME
식용유	6500	맛있는산초유
과자	1800	맛있는맛동산

▼ 문제 풀이 아이디어

❓ **식품분류 별로 가격이 가장 비싼 식품을 찾아, 그 결과를 가격 기준 내림차순으로 정렬하는 SQL 작성하자!**

- `FOOD_PRODUCT` 테이블에서 **식품분류 별**로 **가격이 제일 비싼 식품의 분류, 가격, 이름을 조회**하는 SQL문을 작성해주세요. 이때 **식품분류가 '과자', '국', '김치', '식용유'인 경우만 출력**시켜 주시고 결과는 **식품 가격을 기준으로 내림차순 정렬**해주세요.

- **분류별**(CATEGORY)로 묶어야 함.
- 각 분류에서 가장 비싼 가격(최댓값)을 찾아야 함
- 출력은 **분류, 최고가, 상품명**
- 단, 카테고리는 `'과자', '국', '김치', '식용유'`만
- 최종적으로 **가격 내림차순 정렬**

- **<GROUP BY + MAX>** 문제처럼 보이지만, **상품명까지** 출력해야 하기 때문에 단순 집계만으로는 안 되겠네!

▼ 정답 쿼리

🤓 **FROM → ON → JOIN → WHERE → GROUP BY → 집계함수 → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT**

✅ **1. 윈도우 함수 방식**

- 정답 쿼리

```
SELECT CATEGORY,
      PRICE AS MAX_PRICE,
      PRODUCT_NAME
FROM (
   SELECT CATEGORY,
         PRODUCT_NAME,
         PRICE,
         ROW_NUMBER() OVER (
            PARTITION BY CATEGORY
            ORDER BY PRICE DESC
         ) AS RN
   FROM FOOD_PRODUCT
   WHERE CATEGORY IN ('과자', '국', '김치', '식용유')
) T
WHERE RN = 1
ORDER BY MAX_PRICE DESC;
```

▼ 쿼리 해석

`FOOD_PRODUCT` 테이블에서 **식품분류 별**로 **가격이 제일 비싼 식품의 분류, 가격, 이름을 조회**하는 SQL문을 작성해주세요. 이때 **식품분류가 '과자', '국', '김치', '식용유'인 경우만 출력**시켜 주시고 결과는 **식품 가격을 기준으로 내림차순 정렬**해주세요.

1. FROM 절 서브쿼리 = 인라인 뷰

```
SELECT CATEGORY,
      PRODUCT_NAME,
      PRICE,
      ROW_NUMBER() OVER (
         PARTITION BY CATEGORY
         ORDER BY PRICE DESC
      ) AS RN
FROM FOOD_PRODUCT
WHERE CATEGORY IN ('과자', '국', '김치', '식용유')
```

- `WHERE CATEGORY IN ('과자', '국', '김치', '식용유')`

    - 전체 식품 중에서 과자, 국, 김치, 식용유 카테고리만 조회합니다.
- `ROW_NUMBER() OVER (PARTITION BY CATEGORY ORDER BY PRICE DESC)`
    - 카테고리별로 데이터를 나눈 뒤(`PARTITION BY CATEGORY`) 가격(`PRICE`)을 기준으로 높은 순(`DESC`) 정렬하여 **순위를 매김.**
    - 예를 들어 **식용유**라는 카테고리에 3개 상품이 있으면, 가격이 높은 순서대로 `1, 2, 3` 이라는 번호가 매겨짐.
    - **참고) 왜 `RANK()` 나 `DENSE_RANK()` 가 아닌가?** 🤔
        - `RANK()` → 동점이 있으면 같은 순위(1, 1, 3...)
        - `DENSE_RANK()` → 동점이 있으면 같은 순위지만 다음 순위는 바로 이어짐(1, 1, 2...)
        - `ROW_NUMBER()` → **무조건 1개만 뽑힘** (동점도 강제로 순서가 매겨짐 → 1, 2, 3...)
- 여기서 서브쿼리 결과는 **카테고리별 상품과 그 가격, 그리고 가격 순위를 담은 결과죠.**

👋 윈도우 함수 결과를 필터링하기 위해 인라인뷰를 써서 순위 컬럼을 먼저 만들어두고, 바깥 쿼리에서 조건을 걸어야 하기 때문에 인라인뷰를 사용해볼 수 있었던거죠.

2. 메인 쿼리 부분

```
SELECT CATEGORY,
      PRICE AS MAX_PRICE,
      PRODUCT_NAME
FROM ( ... ) T  -- 서브쿼리
WHERE RN = 1
```

- 서브쿼리 결과에서 `RN = 1`인 행만 선택
- 즉, **각 카테고리에서 가격이 제일 비싼 상품 하나**만 남기고 `PRICE AS MAX_PRICE`로 별칭을 바꿔서, 결과 컬럼 이름을 `MAX_PRICE`로 보여줌.

3. 최종 정렬

```
ORDER BY MAX_PRICE DESC;
```

- 최종적으로 뽑힌 각 카테고리별 최고가 상품들을 **가격 기준 내림차순**으로 정렬

✅ **2. 집계 + JOIN 방식**

- 과자, 국, 김치, 식용유 카테고리에서 각각 최고 가격을 가진 상품 하나를 찾아서, 전체 결과를 가격 내림차순으로 정렬해 출력한다.
- 정답 쿼리

```
SELECT FP.CATEGORY,
      FP.PRICE AS MAX_PRICE,
      FP.PRODUCT_NAME
FROM FOOD_PRODUCT FP
JOIN (
   SELECT CATEGORY, MAX(PRICE) AS MAX_PRICE
   FROM FOOD_PRODUCT
   WHERE CATEGORY IN ('과자', '국', '김치', '식용유')
   GROUP BY CATEGORY
```

```
) TMP
  ON FP.CATEGORY = TMP.CATEGORY
 AND FP.PRICE = TMP.MAX_PRICE
WHERE FP.CATEGORY IN ('과자', '국', '김치', '식용유')
ORDER BY FP.PRICE DESC;
```

▼ 쿼리 해석

`FOOD_PRODUCT` 테이블에서 **식품분류 별**로 **가격이 제일 비싼 식품의 분류, 가격, 이름을 조회**하는 SQL문을 작성해주세요. 이때 **식품분류가 '과자', '국', '김치', '식용유'인 경우만 출력**시켜 주시고 결과는 **식품 가격을 기준으로 내림차순 정렬**해주세요.

1. 내부 서브쿼리 (`TMP`)

```
SELECT CATEGORY, MAX(PRICE) AS MAX_PRICE
FROM FOOD_PRODUCT
WHERE CATEGORY IN ('과자', '국', '김치', '식용유')
GROUP BY CATEGORY
```

- `FOOD_PRODUCT` 테이블에서 **카테고리별(CATEGORY)** 로 그룹화
- 각 그룹에서 `PRICE` (가격)의 최댓값(`MAX(PRICE)`)을 구한다.
- `WHERE` 절로 `'과자', '국', '김치', '식용유'` 카테고리만 필터링
- 결과는? 카테고리별로 **최고가만 남은 집계 결과!**
- 결과 예시)

CATEGORY	MAX_PRICE
식용유	6500
과자	1800
국	5200
김치	4700

2. JOIN (`FP JOIN TMP`)

```
FROM FOOD_PRODUCT FP
JOIN (
    SELECT CATEGORY, MAX(PRICE) AS MAX_PRICE
    FROM FOOD_PRODUCT
    WHERE CATEGORY IN ('과자', '국', '김치', '식용유')
    GROUP BY CATEGORY
) TMP
ON FP.CATEGORY = TMP.CATEGORY
AND FP.PRICE = TMP.MAX_PRICE
```

- 원본 테이블 `FOOD_PRODUCT` 를 `FP` 라는 별칭으로 사용
- 서브쿼리 결과 `TMP` (임시테이블)과 조인
- 조인 조건
  - **카테고리가 같고** (`FP.CATEGORY = TMP.CATEGORY`)
  - **해당 카테고리의 최고 가격과 일치** (`FP.PRICE = TMP.MAX_PRICE`)
- 결과는? 각 카테고리에서 가격이 가장 비싼 상품만 필터링됨
- 결과 예시)

CATEGORY	PRODUCT_ID	PRODUCT_NAME	PRICE
식용유	P0020	맛있는산초유	6500
과자	P0099	맛있는맛동산	1800
국	P0100	맛있는김치찌개	5200
김치	P0101	맛있는배추김치	4700

3. 최종 SELECT & ORDER

```
SELECT FP.CATEGORY,
       FP.PRICE AS MAX_PRICE,
       FP.PRODUCT_NAME
WHERE FP.CATEGORY IN ('과자', '국', '김치', '식용유')
ORDER BY FP.PRICE DESC;
```

- 출력 컬럼: `CATEGORY`, `MAX_PRICE`, `PRODUCT_NAME`
- 카테고리를 다시 한 번 `'과자', '국', '김치', '식용유'` 로 제한
- 가격(`PRICE`)을 기준으로 **내림차순** 정렬
- 최종 실행 결과)

CATEGORY	MAX_PRICE	PRODUCT_NAME
식용유	6500	맛있는산초유
국	5200	맛있는김치찌개
김치	4700	맛있는배추김치
과자	1800	맛있는맛동산