DATA AGGREGATION

***
ORDER:
1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY
***

***
EXAMPLE TO HAVE 2 OPTIONS FOR FILLING 1 COLUMN DEPENDING ON THE DATA INSIDE:

```
SELECT
      CONCAT(first_name, ' ', last_name) AS full_name,
      COALESCE(AGE(died, born)::VARCHAR(20), CONCAT('Alive: ', AGE(now(),
born)::VARCHAR(30))) AS lifespan
FROM
      authors;
```
***

1.Grouping: consolidating data based on criteria
- Grouping allows taking data into separate groups based on a common property

```
SELECT column_one,
         column_two
FROM table_name

GROUP BY column_one,
             column_two;
```

2. Aggregate Functions:
- Used to operate over one or more groups performing data analysis on every one
- MIN, MAX, AVG, COUNT, SUM, etc.
- They usually ignore NULL values

```
SELECT column_one,
         aggregate_function(column_two)
FROM table_name
GROUP BY column_one;
```

- COUNT - counts the values(NOT NULLS) in one or more columns based on grouping criteria

```
SELECT
      "department_id",
      COUNT("id") AS "employee_count"
FROM "employees"
GROUP BY "department_id"
ORDER BY "department_id";
```

PROBLEM: COUNT will ignore every employee with NULL value for salary
IN ORDER TO AVOID USE:

```
SELECT
      "department_id",
      COUNT(*) AS "Employees Count"
```

```
      FROM employees
      GROUP BY department_id,
      ORDER BY department_id ASC;



      SELECT
            "department_id"
            COUNT("salary") AS "employee_count"
      FROM "employees"
      GROUP BY "department_id"
      ORDER BY "department_id";
```

- SUM - sums the values in a column based on grouping criteria

      PROBLEM: if all employees in a department have no salaries, NULL will be displayed

```
      SELECT "department_id"
            SUM("salary") AS "total_salaries"
      FROM "employees"
      GROUP BY "department_id"
      ORDER BY "department_id";
```

- MAX - takes the maximum value in a column

```
      SELECT "department_id"
            MAX("salary") AS "max_salary"
      FROM "employees"
      GROUP BY "department_id"
      ORDER BY "department_id"
```

- MIN - takes the minimum value in a column

```
      SELECT "department_id"
            MIN("salary") AS "min_salary"
      FROM "employees"
      GROUP BY "department_id"
      ORDER BY "department_id"
```

AVG - calculates the average value in a column

```
      SELECT "department_id"
            AVG("salary") AS "average_salary"
      FROM "employees"
      GROUP BY "department_id"
      ORDER BY "department_id"
```

3. Having - using predicates while grouping
      - The HAVING clause is used to filter data based on aggregate values
      - We cannot use it without grouping before that
      - Any aggregate functions in the "HAVING" clause and in the "SELECT" statement are executed one time only
      - Unlike HAVING, the WHERE clause filters rows before the aggregation

      WHERE is used for row-level filtering before grouping or aggregation,
      while HAVING is used for filtering groups based on aggregate results after grouping and aggregation.

```
SELECT "department_id"
      SUM("salary") AS "Total Salary"
FROM "employees"
GROUP BY "department_id"
HAVING SUM("salary") < 4200
ORDER BY "department_id"
```

4. Conditions - creating conditional queries - CASE Expression
   - We can check if a condition(case) is true or false
   - Then we can proceed, depending on the result

   - The PostgreSQL CASE expression is the same as IF/ELSE segment in other programming language
   - It allows you to add if-else logic from a powerful query
   - the CASE expression has two forms:

GENERAL
      AND
SIMPLE
      FORM

   - Can be used in SELECT, WHERE, GROUP BY clauses

```
CASE
      WHEN condition_1 THEN result_1
      WHEN condition_2 THEN result 2
      ...
[ELSE else_result]
END AS column_name
```

```
SELECT id, first_name, last_name, salary
      CASE
            WHEN department_id = 1 THEN 'Management'
            WHEN department_id = 2 THEN 'Kitchen Staff'
            WHEN department_id = 3 THEN 'Service Staff'
      ELSE 'Other'
      END AS department_name
FROM employees;
```

```
CASE expression
      WHEN value_1 THEN result_1
      WHEN value_2 THEN result_2
      ...
ELSE result_n
END AS column_name

SELECT
      id,
      first_name,
      last_name,
      TRUNC(salary, 2) as salary,
      department_id,

      CASE department_id
            WHEN 1 THEN 'Management'
            WHEN 2 THEN 'Kitchen Staff'
            WHEN 3 THEN 'Service Staff'
      ELSE 'Other'
```

```
      END AS department_name
FROM employees
ORDER BY id;


SELECT
      SUM(salary) AS total_salaries,
      SUM(CASE department_id
                  WHEN 1 THEN salary*1.15
                  WHEN 2 THEN salary*1.10
            ELSE salary*1.05
      END) AS total_increased_salaries
FROM employees;


SELECT
      CASE
            WHEN salary < 1000 THEN 'LOW'
            WHEN salary <=3000 THEN 'Middle'
      ELSE 'High'
      END AS "salary_range"
      COUNT(salary) AS "salary_count"
FROM employees
GROUP BY salary_range
HAVING CASE COUNT(salary)
            WHEN 0 THEN 'false'::boolean
      ELSE 'true'::boolean
      END;
```