# Aggregation in SQL

## SQL and aggregation

- sometimes instead of listing a list of rows, we prefer to find out their number / sum and etc.
  - This is what the so-called aggregation functions are used for:
  - Sum, Min, Max, Avg, Stdev, Count, ...
  - https://www.postgresql.org/docs/current/static/functions-aggregate.html

- Sometimes we want to group rows by some attribute
  - e.g. we want to group employees according to their department, etc.
  - or we want the number of employees in particular departments

## GROUP BY

- **SELECT** <list of attributes>
  **FROM** <list of tables>
  **WHERE** <condition>
  **GROUP BY** <list of attributes>
  **HAVING** <condition>

- GROUP BY - groups rows with the same value in the listed attributes
- for each group, there will be 1 line in the output
- the list of attributes after SELECT can only contain attributes listed after GROUP BY and aggregate functions
  - This is not entirely true for all database systems (e.g. MySQL does not have such a restriction)
- a condition in HAVING can contain aggregate functions, while WHERE cannot

## GROUP BY example

- **SELECT** deptno, COUNT(*) as c
  **FROM** emp
  **WHERE** sal > 3000
  **GROUP BY** deptno
  **HAVING** COUNT(*)>=2

- note that we need to write COUNT(*) twice

# GROUP BY example

| name | deptno | sal |
|---|---|---|
| John | 10 | 1000 |
| Thomas | 10 | 3100 |
| George | 10 | 3200 |
| Lucas | 20 | 3100 |
| Bob | 20 | 2050 |
| Joe | 30 | 1000 |
| Francis | 30 | 3050 |
| Hugo | 40 | 1000 |
| Mike | 40 | 5000 |
| Robert | 40 | 2900 |
| Anna | 50 | 8000 |

WHER
E

| name | deptno | sal |
|---|---|---|
| George | 10 | 3200 |
| Thomas | 10 | 3100 |
| Lucas | 20 | 3100 |
| Francis | 30 | 3050 |
| Mike | 40 | 5000 |
| Anna | 50 | 8000 |

```
SELECT deptno, COUNT(*)
FROM emp
WHERE sal > 3000
GROUP BY deptno
HAVING COUNT(*)>=2
```

# GROUP BY example

| name | deptno | sal |
|---|---|---|
| George | 10 | 3200 |
| Thomas | 10 | 3100 |
| Lucas | 20 | 3100 |
| Francis | 30 | 3050 |
| Mike | 40 | 5000 |
| Anna | 50 | 8000 |

GROUP BY

**SELECT** deptno, COUNT(*)
**FROM** emp
**WHERE** sal > 3000
**GROUP BY** deptno
**HAVING** COUNT(*)>=2

| deptno | COUNT(*) | Name | deptno | sal |
|---|---|---|---|---|
| 10 | 2 | | | |
| | | George | 10 | 3200 |
| | | Thomas | 10 | 3100 |
| 20 | 1 | | | |
| | | Lucas | 20 | 3100 |
| 30 | 1 | | | |
| | | Francis | 30 | 3050 |
| 40 | 1 | | | |
| | | Mike | 40 | 5000 |
| 50 | 1 | | | |
| | | Anna | 50 | 8000 |

# GROUP BY example

| deptno | count(*) | Name | deptn | sal |
|---|---|---|---|---|
| 10 | 2 | | | |
| | | George | 10 | 3200 |
| | | Thomas | 10 | 3100 |
| 20 | 1 | | | |
| | | Lucas | 20 | 3100 |
| 30 | 1 | | | |
| | | Francis | 30 | 3050 |
| 40 | 1 | | | |
| | | Mike | 40 | 5000 |
| 50 | 1 | | | |
| | | Anna | 50 | 8000 |

$\xrightarrow{\text{HAVING}}$

| deptno | count(*) |
|---|---|
| 10 | 2 |

**SELECT** deptno, COUNT(*)
**FROM** emp
**WHERE** sal > 3000
**GROUP BY** deptno
**HAVING** COUNT(*)>=2

# Other aspects

- the list of attributes after SELECT can only contain the attributes listed after GROUP BY and aggregate functions
- This is a bit annoying for programmers:
  - student(StudentID, Meno, Priezvisko, TriedaID)
    trieda(TriedaID, Nazov)
  - **SELECT s.classid, c.name, COUNT(*)**
  - **FROM student as s, class as c**
  - **WHERE  s.classid = c.classid GROUP BY s.classid, c.name**
  - ClassID clearly specifies the name of the class, but the programmer has to write it unnecessarily 2x

## Other aspects

- MySQL:

  In the SELECT section, we can use any attribute. If it is from a set of attributes that are not in GROUP BY, a random element from the group is selected

- PostgreSQL:

  When GROUP BY is present, or any aggregate functions are present, it is not valid for the SELECT list expressions to refer to ungrouped columns except within aggregate functions or when the ungrouped column is functionally dependent on the grouped columns, since there would otherwise be more than one possible value to return for an ungrouped column. A functional dependency exists if the grouped columns (or a subset thereof) are the primary key of the table containing the ungrouped column.

# Other aspects

- If SELECT contains an aggregate function, but without GROUP BY
  - then all the rows seem to be included in one group – the output is a row, e.g.:
  - SELECT COUNT(*) FROM emp;
  - SELECT MAX(sal) FROM emp;

  - similarly, if we state HAVING without GROUP BY
  - (avoid this one, it's confusing and, unlike the previous one, useless)-

## Other aspects

- beware of NULL in aggregate functions
- some aggregate functions omit rows with NULL
- e.g. the SUM for 1 + NULL is 1, while the expression 1 + NULL is NULL
- moreover, if all rows are NULL, then the result is NULL

## Other aspects

- if we need to find values for which e.g. the maximum (arg max) is reached, it cannot be written in SQL with one query, we need a nested query:

```
SELECT name
FROM emp
WHERE salary = (SELECT MAX(salary) FROM emp);
```

- The database system automatically converts the result of a nested selection (a session with 1 column and 1 row) to a number
- Try what happens if the EMP table contains no records