SQL essentials

Mastering database queries:
a comprehensive guide to SQL

Learning session 05

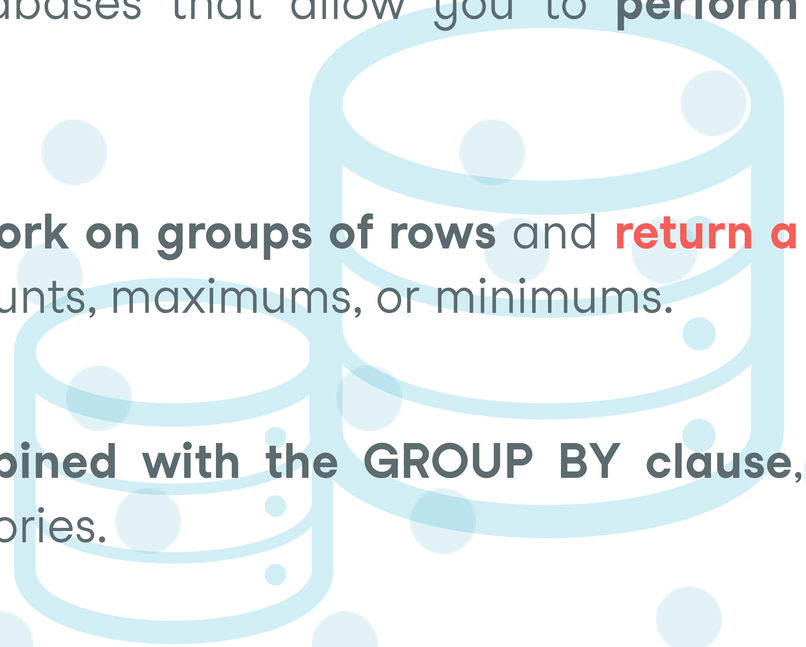
Aggregate functions and grouping

Instructor
Péter Fülöp
peteristvan.fulop@hcl.com

Understanding aggregate functions

Understanding Aggregate Functions in SQL

- **Aggregate functions** are specialized operations in SQL databases that allow you to **perform calculations and data summarization on sets of values**.
- Instead of operating on individual rows, **aggregate functions work on groups of rows** and **return a single result**, typically providing insights like sums, averages, counts, maximums, or minimums.
- These aggregate functions are particularly useful when **combined with the GROUP BY clause**, allowing you to aggregate data within specific groups or categories.
- They are **essential tools for data analysis** and reporting in SQL.



Aggregate Functions in SQL

- COUNT
- SUM
- AVG
- MIN
- MAX

Understanding the COUNT Aggregate Function in SQL

- The `COUNT` function is a basic SQL aggregate function **used to count the number of rows** in a database table or set/group of rows.
- `COUNT(DISTINCT column_name)` is used to count only the unique values in a column.
- `COUNT(column_name)` does not include NULL values in its count, whereas COUNT(*) counts all rows, regardless of NULL values.

Count the number of employees in the employees table.

SQL

```sql
SELECT COUNT(*)
FROM employees;
```

Calculate the total number of unique bands in the employees table, excluding any employees whose band_id is NULL.

SQL

```sql
SELECT COUNT(DISTINCT band_id)
FROM employees;
```

Grasping COUNT function's approach to NULL values

Display the preferred language IDs present in the employees table.

SQL

```sql
SELECT preferred_language_id
FROM employees;
```

OUTPUT

	preferred_language_id
1	<null>
2	<null>
3	<null>
4	893
5	367

29 rows

Calculate the total number of preferred languages found in the employees table.

SQL

```sql
SELECT COUNT(preferred_language_id)
FROM employees;
```

OUTPUT

	count
1	24

1 row

Grasping COUNT function's approach to NULL values

Display the unique preferred language IDs present in the employees table.

SQL

```sql
SELECT DISTINCT preferred_language_id
FROM employees;
```

OUTPUT

	preferred_language_id
1	<null>
2	894
3	369
4	893
5	367

5 rows

Calculate the total number of unique preferred languages found in the employees table.

SQL

```sql
SELECT COUNT(DISTINCT preferred_language_id)
FROM employees;
```

OUTPUT

	count
1	4

1 row

Working with the SUM functions

Calculate the sum of monthly salaries that the company is required to pay to its employees, as recorded in the employees table.

SQL

```sql
SELECT
      SUM(salary) as monthly_salaries_total
FROM
      employees;
```

OUTPUT

	monthly_salaries_total
1	92100

1 rows

Working with MIN, MAX, AVG functions

Select the minimum, maximum, and average age of employees from the employees table.

SQL

```sql
SELECT
    EXTRACT(YEAR FROM MIN(age(birth_date))) AS min_age,
    EXTRACT(YEAR FROM MAX(age(birth_date))) AS max_age,
    EXTRACT(YEAR FROM AVG(age(birth_date))) AS avg_age
FROM employees;
```

OUTPUT

	min_age	max_age	avg_age
1	25	42	31

1 rows

The GROUP BY clause

The GROUP BY clause

- The **GROUP BY** clause in SQL is a powerful tool for **organizing data into groups based on common attributes**. This clause is particularly useful when you need to perform aggregate functions, like calculating sums, averages, or counts, on distinct categories within your data.

- The true power of GROUP BY emerges when it's used in conjunction with aggregate functions like COUNT(), SUM(), AVG(), MAX(), and MIN(). **These functions compute a single result from a group of input values.**

- The **HAVING clause** is used to filter groups created by GROUP BY based on a specified condition. It's important to note that **HAVING is different from WHERE**:
  WHERE filters rows before the grouping occurs, while HAVING filters groups after the grouping has been applied.

Using the GROUP BY clause

Show the lowest, highest, and mean salary for each band. Include the band name, label the average salary as avg_sal, the highest salary as max_sal, and the lowest salary as min_sal. Omit employees who are not assigned to any band.

SQL

```sql
-- First step: focus on grouping
SELECT MAX(salary) as max_sal,
       MIN(salary) as min_sal,
       AVG(salary) as avg_sal
FROM employees
GROUP BY band_id;
```

OUTPUT

	band_id	max_sal	min_sal	avg_sal
1	<null>	3200	3200	3200
2	4	4500	3000	3533.3333333333333333
3	6	3800	2600	3175
4	2	4000	2900	3460
5	7	3800	2900	3300

8 rows

Using the GROUP BY clause

Let's exclude the employees who haven't been assigned a band.

SQL

```
SELECT band_id,
       MAX(salary) as max_sal,
       MIN(salary) as min_sal,
       AVG(salary) as avg_sal
FROM employees
WHERE band_id IS NOT NULL
GROUP BY band_id;
```

OUTPUT

	band_id	max_sal	min_sal	avg_sal
1	4	4500	3000	3533.3333333333333
2	6	3800	2600	3175
3	2	4000	2900	3460
4	7	3800	2000	2900

7 rows

Using the GROUP BY clause

Let's now display the band name by joining with the bands table.

SQL

```sql
SELECT e.band_id, b.name as band_name,
       MAX(e.salary) as max_sal,
       MIN(e.salary) as min_sal,
       AVG(e.salary) as avg_sal
FROM employees e
INNER JOIN bands b ON e.band_id=b.id
WHERE e.band_id IS NOT NULL
GROUP BY e.band_id, b.name;
```

OUTPUT

band_id	band_name	max_sal	min_sal	avg_sal
3	Mid-Senior	3500	2000	2733.3333333333333
5	Senior 2	4500	2600	3500
1	Junior 1	3500	2500	3066.6666666666667
4	Senior 1	4500	3000	3533.3333333333333

7 rows

Using the GROUP BY clause

Let's explore alternative solutions that might be more efficient in certain scenarios by utilizing our understanding of derived tables and Common Table Expressions (CTEs).

SQL

```sql
SELECT
    bands.id, bands.name as band_name,
    stats.max_sal, stats.min_sal,
    stats.avg_sal
FROM (SELECT band_id,
             MAX(salary) as max_sal,
             MIN(salary) as min_sal,
             AVG(salary) as avg_sal
      FROM employees
      WHERE band_id IS NOT NULL
      GROUP BY band_id) AS stats
      INNER JOIN bands
      ON stats.band_id = bands.id;
```

SQL

```sql
WITH stats AS (SELECT band_id,
                      MAX(salary) as max_sal,
                      MIN(salary) as min_sal,
                      AVG(salary) as avg_sal
               FROM employees
               WHERE band_id IS NOT NULL
               GROUP BY band_id)
SELECT bands.id,
       bands.name as band_name,
       stats.max_sal,
       stats.min_sal,
       stats.avg_sal
FROM stats
     INNER JOIN bands
     ON stats.band_id = bands.id;
```

Filtering the groups using the HAVING clause

Let's assume we want to omit bands where the average salary is 3000 or lower. Additionally, we'll present the average salary rounded to two decimal places.

SQL

```sql
WITH stats AS (SELECT band_id,
                      MAX(salary) as max_sal,
                      MIN(salary) as min_sal,
                      ROUND(AVG(salary),2) as avg_sal
               FROM employees
               WHERE band_id IS NOT NULL
               GROUP BY band_id
               HAVING AVG(salary) > 3000)
SELECT bands.id,
       bands.name as band_name,
       stats.max_sal,
       stats.min_sal,
       stats.avg_sal
FROM stats
     INNER JOIN bands
     ON stats.band_id = bands.id;
```

	band_id	band_name	max_sal	min_sal	avg_sal
1	6	Senior 3	3800	2600	3175
2	5	Senior 2	4500	2600	3500
3	4	Senior 1	4500	3000	3533.33
4	2	Junior 2	4000	2900	3460
5	1	Junior 1	3500		

5 rows

Example 1: using the GROUP BY clause

Determine the average age corresponding to each job title, displaying two columns: the Job Title and the Average Age. Sort the results in ascending order based on the average age.

SQL

```sql
WITH age_stats
AS (SELECT job_title_id,
           EXTRACT(YEAR FROM AVG(age(birth_date)))
               AS avg_age
    FROM employees
    WHERE job_title_id IS NOT NULL
    GROUP BY job_title_id)

SELECT name as job_title, age_stats.avg_age
FROM job_titles
         INNER JOIN age_stats
             ON job_titles.id = age_stats.job_title_id
ORDER BY age_stats.avg_age ASC;
```

	job_title	avg_age
1	Manager	28
2	Data engineer	30
3	BI Support specialist	30
4	BI developer	31
5	Data Scientist	31

5 rows

Example 2: using the GROUP BY clause

Obtain a list of countries where more than one language is spoken, along with the total number of different languages spoken in each of these countries.

SQL

```sql
SELECT c.name as country, lang_stats.num_languages
FROM countries c
         INNER JOIN (SELECT
                         code,
                         COUNT(*) as num_languages
                     FROM languages
                     GROUP BY code
                     HAVING COUNT(*) > 1) lang_stats
         ON c.code = lang_stats.code
ORDER BY lang_stats.num_languages DESC;
```

2	Zimbabwe	16
3	Ethiopia	16
4	Nepal	14
5	India	14
6	France	13
7	South Africa	13
8	Mali	

184 rows

Example 3: using the GROUP BY clause (same as assignment 4.3)

Retrieve the full names, band designations, and salaries of employees who have the top salary within their respective bands.

SQL

```sql
SELECT e.first_name || ' ' || e.last_name as full_name,
       b.name as band_name, e.salary
FROM employees e
         INNER JOIN bands b ON e.band_id = b.id
         INNER JOIN (SELECT band_id, MAX(salary) max_sal
                     FROM employees
                     WHERE band_id IS NOT NULL
                     GROUP BY band_id) AS top_salaries
                    ON e.band_id = top_salaries.band_id
                    AND e.salary = top_salaries.max_sal;
```

	full_name	band_name	salary
1	Anna Kovacs	Junior 1	3500
2	Balazs Toth	Junior 2	4000
3	Istvan Horvath	Senior 2	4500
4	Gabor Kovacs	Mid-Senior	3500
5	Anna Nagy	Senior 3	3800
6	Eva Horvath	Senior 1	4500
7	Anita Molnar	Senior 4	

7 rows