6. Grouping and Filtering Data

In SQL, grouping and filtering data are essential operations for analyzing datasets. The `GROUP BY` clause allows you to aggregate data into summary rows, while the `HAVING` clause enables you to filter these grouped results based on specific conditions.

We'll explore these concepts in detail using the `Employees` and `Departments` tables, accompanied by several practical examples.

Tables Overview

`Employees` Table

The `Employees` table contains information about employees, including their department, salary, and join date.

EmployeeID	FirstName	LastName	Department	Salary	JoinDate	ManagerID
1	John	Doe	HR	50000.00	2020-01-15	NULL
2	Jane	Smith	IT	60000.00	2019-03-10	NULL
3	Michael	Johnson	Finance	75000.00	2021-07-22	NULL
4	Emily	Davis	IT	65000.00	2022-11-11	2
5	James	Brown	HR	45000.00	2018-05-30	2
6	Robert	Wilson	Marketing	55000.00	2023-04-18	2
7	Sarah	Taylor	HR	62000.00	2020-08-15	3
8	David	Anderson	Finance	54000.00	2023-01-20	3
9	Laura	Martinez	IT	58000.00	2021-02-12	3
10	Emily	Clark	Sales	47000.00	2019-11-05	NULL
11	Anna	Lee	Support	49000.00	2022-03-18	NULL
12	Paul	Walker	Marketing	51000.00	2021-08-24	NULL
13	Nina	Scott	Legal	72000.00	2022-12-15	NULL
14	Tom	Moore	Operations	56000.00	2023-02-01	NULL
15	Olivia	Martin	Development	65000.00	2023-07-10	NULL
16	Liam	Taylor	HR	53000.00	2018-06-25	NULL
17	Sophia	Green	Finance	76000.00	2019-09-19	NULL
18	Jackson	Harris	IT	67000.00	2020-12-05	NULL
19	Ava	Roberts	Sales	48000.00	2021-01-10	NULL
20	Mason	Carter	Support	50000.00	2022-05-16	NULL
21	Isabella	Morris	Marketing	54000.00	2022-11-30	NULL
22	Ethan	Walker	Legal	73000.00	2023-03-22	NULL
23	Mia	Allen	Operations	57000.00	2023-06-15	NULL
24	Jacob	Young	Development	66000.00	2023-01-08	NULL
25	Harper	King	Research	68000.00	2023-04-05	NULL
26	Benjamin	Adams	Sales	49000.00	2021-05-20	NULL
27	Charlotte	Baker	Support	51000.00	2022-07-12	NULL
28	Alexander	Nelson	Legal	74000.00	2023-08-18	NULL
29	Ella	Cruz	Development	67000.00	2023-10-01	NULL
30	William	Green	Research	69000.00	2023-09-15	NULL

`Departments` Table

The `Departments` table contains information about various departments and their locations.

DepartmentID	DepartmentName	Location
1	HR	New York
2	Finance	London
3	IT	San Francisco
4	Marketing	Chicago
5	Sales	Boston
6	Support	Seattle
7	Legal	Houston
8	Operations	Phoenix
9	Development	Austin
10	Research	Denver
11	IT	San Jose
12	HR	Los Angeles
13	Finance	Paris
14	Marketing	Dallas
15	Sales	San Diego
16	Support	San Francisco
17	Legal	Philadelphia
18	Operations	San Antonio
19	Development	San Jose
20	Research	Chicago
21	HR	Miami
22	Finance	New York
23	IT	Seattle
24	Marketing	Boston
25	Sales	Austin
26	Support	Phoenix
27	Legal	Denver
28	Operations	Philadelphia
29	Development	Houston
30	Research	San Antonio

GROUP BY Clause

Theory

The `GROUP BY` clause is a powerful tool in SQL used to organize data into distinct groups. By grouping rows that have identical values in specified columns, you can apply aggregate functions like `COUNT`, `SUM`, `AVG`, `MAX`,

and `MIN` to these groups. This allows for concise data summaries and is particularly useful for reporting and data analysis.

Practical Use Cases

- **Business Reporting:** Summarize sales data by region, product, or sales representative.
- **Financial Analysis:** Calculate total expenditures per department or monthly revenue.
- **Human Resources:** Group employees by department to calculate average salary, count of employees, or total department payroll.

Example 1: Grouping Employees by Department

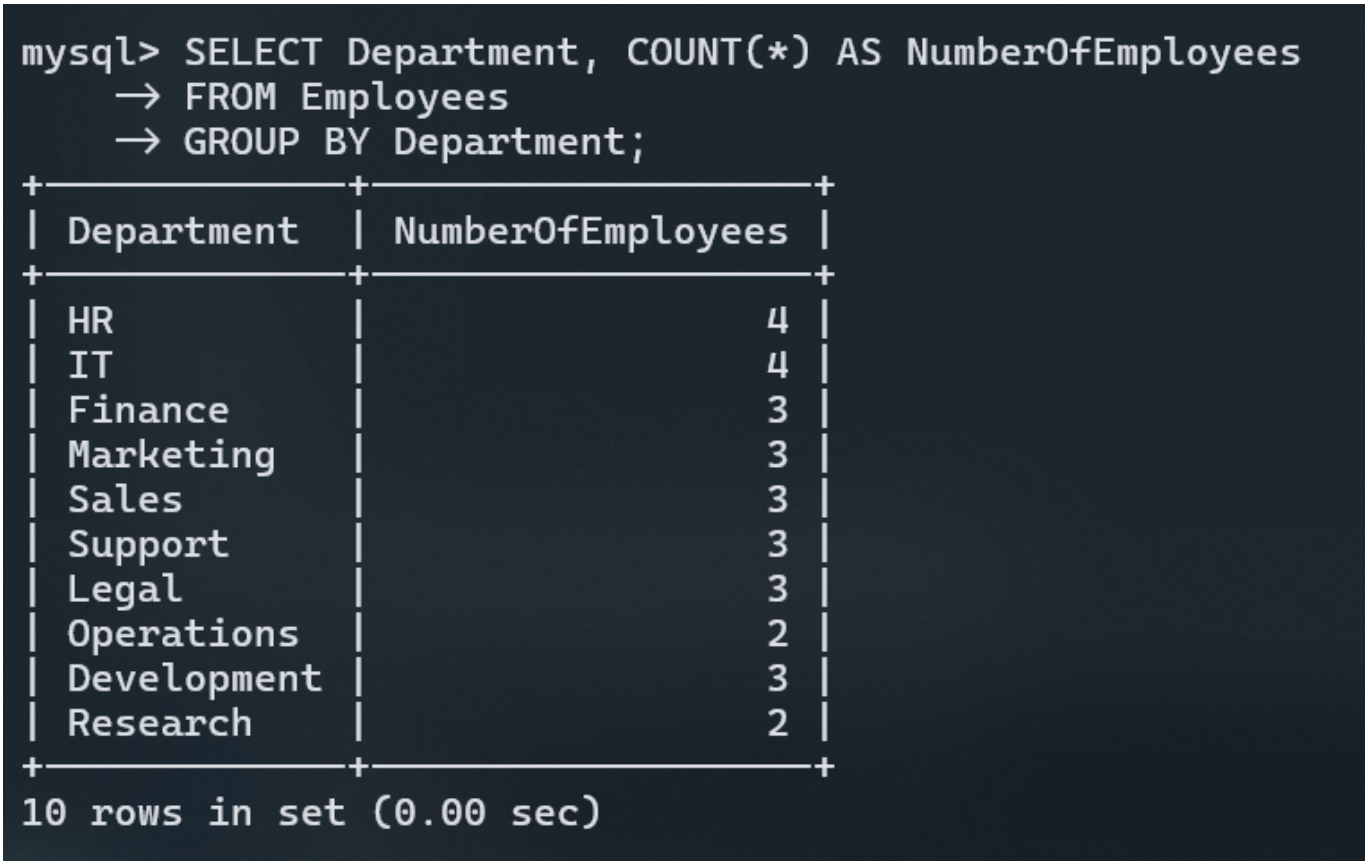
Let's begin by counting the number of employees in each department.

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department;
```

Explanation:

- **SELECT Department:** We are grouping the data based on the `Department` column.
- **COUNT(*):** Counts the number of employees in each department.
- **GROUP BY Department:** Groups the rows by the `Department` column.

Result:



Example 2: Calculating Total Salary by Department

To understand how much each department spends on salaries, we can sum up the salaries of all employees within each department.

```
SELECT Department, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Department;
```

Explanation:

- **SUM(Salary):** Sums the salaries of all employees in each department.
- **GROUP BY Department:** Groups the results by the `Department` column.

Result:

```
mysql> SELECT Department, SUM(Salary) AS TotalSalary
    -> FROM Employees
    -> GROUP BY Department;
+-------------+-------------+
| Department  | TotalSalary |
+-------------+-------------+
| HR          |   210000.00 |
| IT          |   250000.00 |
| Finance     |   205000.00 |
| Marketing   |   160000.00 |
| Sales       |   144000.00 |
| Support     |   150000.00 |
| Legal       |   219000.00 |
| Operations  |   113000.00 |
| Development |   198000.00 |
| Research    |   137000.00 |
+-------------+-------------+
10 rows in set (0.00 sec)
```

Example 3: Finding the Maximum Salary in Each Department

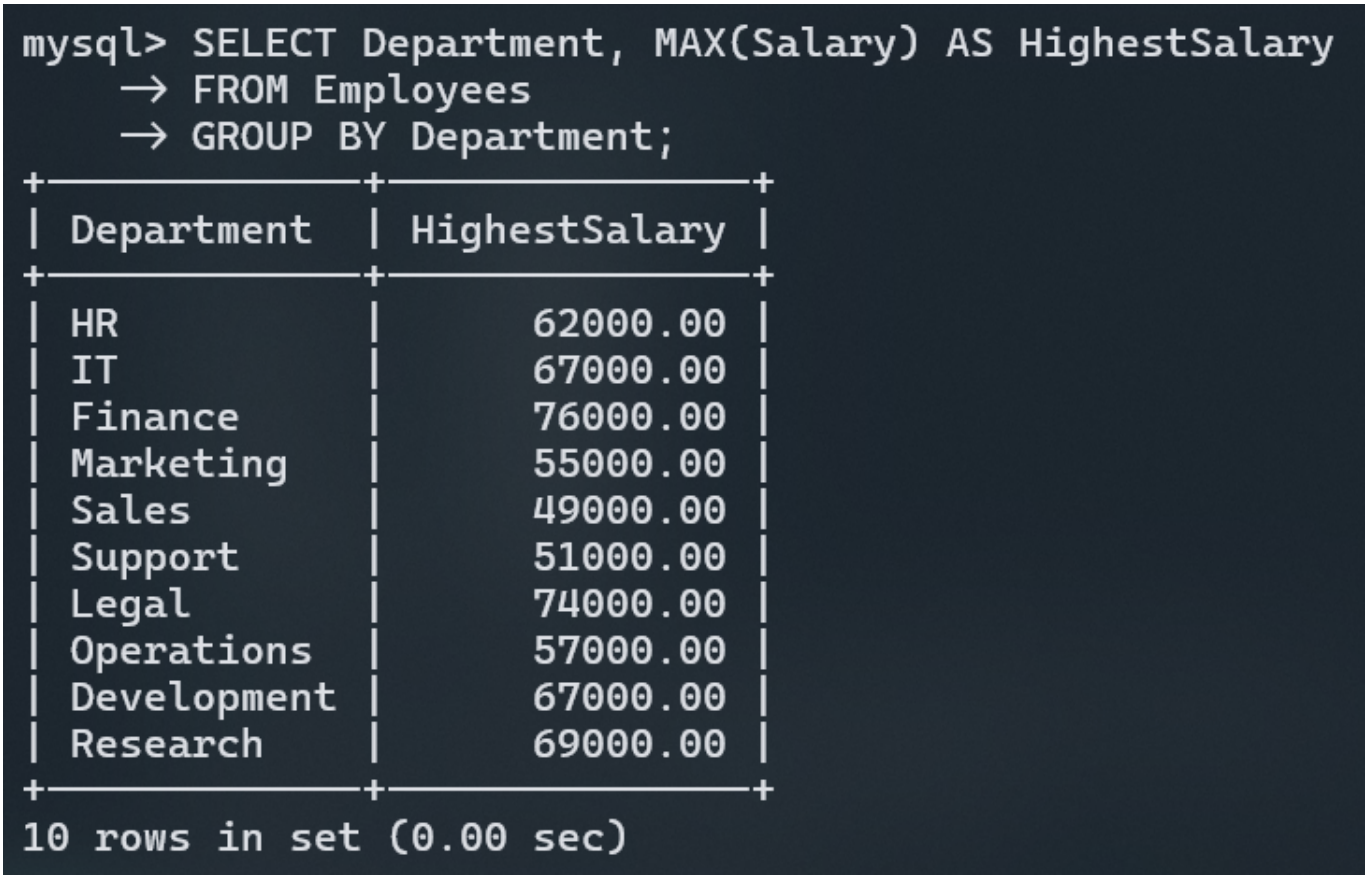
We can determine the highest salary in each department.

```
SELECT Department, MAX(Salary) AS HighestSalary
FROM Employees
GROUP BY Department;
```

Explanation:

- **MAX(Salary):** Finds the maximum salary in each department.
- **GROUP BY Department:** Groups the results by department.

Result:



Example 4: Grouping Employees by Department and Join Year

To analyze the number of employees who joined each department in a particular year, we can use the `YEAR()` function to extract the year from the `JoinDate` column.

```
SELECT Department, YEAR(JoinDate) AS JoinYear, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department, YEAR(JoinDate);
```

Explanation:

- **YEAR(JoinDate):** Extracts the year from the `JoinDate` column.
- **GROUP BY Department, YEAR(JoinDate):** Groups the results by department and the year of joining.

Result:

```
mysql> SELECT Department, YEAR(JoinDate) AS JoinYear, COUNT(*) AS NumberOfEmployees
    -> FROM Employees
    -> GROUP BY Department, YEAR(JoinDate);
+-------------+----------+---------------------+
| Department  | JoinYear | NumberOfEmployees |
+-------------+----------+---------------------+
| HR          |     2020 |                   2 |
| IT          |     2019 |                   1 |
| Finance     |     2021 |                   1 |
| IT          |     2022 |                   1 |
| HR          |     2018 |                   2 |
| Marketing   |     2023 |                   1 |
| Finance     |     2023 |                   1 |
| IT          |     2021 |                   1 |
| Sales       |     2019 |                   1 |
| Support     |     2022 |                   3 |
| Marketing   |     2021 |                   1 |
| Legal       |     2022 |                   1 |
| Operations  |     2023 |                   2 |
| Development |     2023 |                   3 |
| Finance     |     2019 |                   1 |
| IT          |     2020 |                   1 |
| Sales       |     2021 |                   2 |
| Marketing   |     2022 |                   1 |
| Legal       |     2023 |                   2 |
| Research    |     2023 |                   2 |
+-------------+----------+---------------------+
20 rows in set (0.00 sec)
```

HAVING Clause

Theory

The `HAVING` clause allows you to filter groups of data created by the `GROUP BY` clause. While the `WHERE` clause filters rows before they are grouped, `HAVING` filters the groups themselves based on aggregate functions. This is especially useful for refining your results after performing calculations like averages, totals, and counts.

Practical Use Cases

- **Financial Compliance:** Filter departments with average salaries below a certain threshold.
- **Sales Analysis:** Identify sales teams that have underperformed based on total sales.
- **Operational Efficiency:** Highlight departments with fewer than a specified number of employees.

Example 1: Filtering Departments with More Than 2 Employees

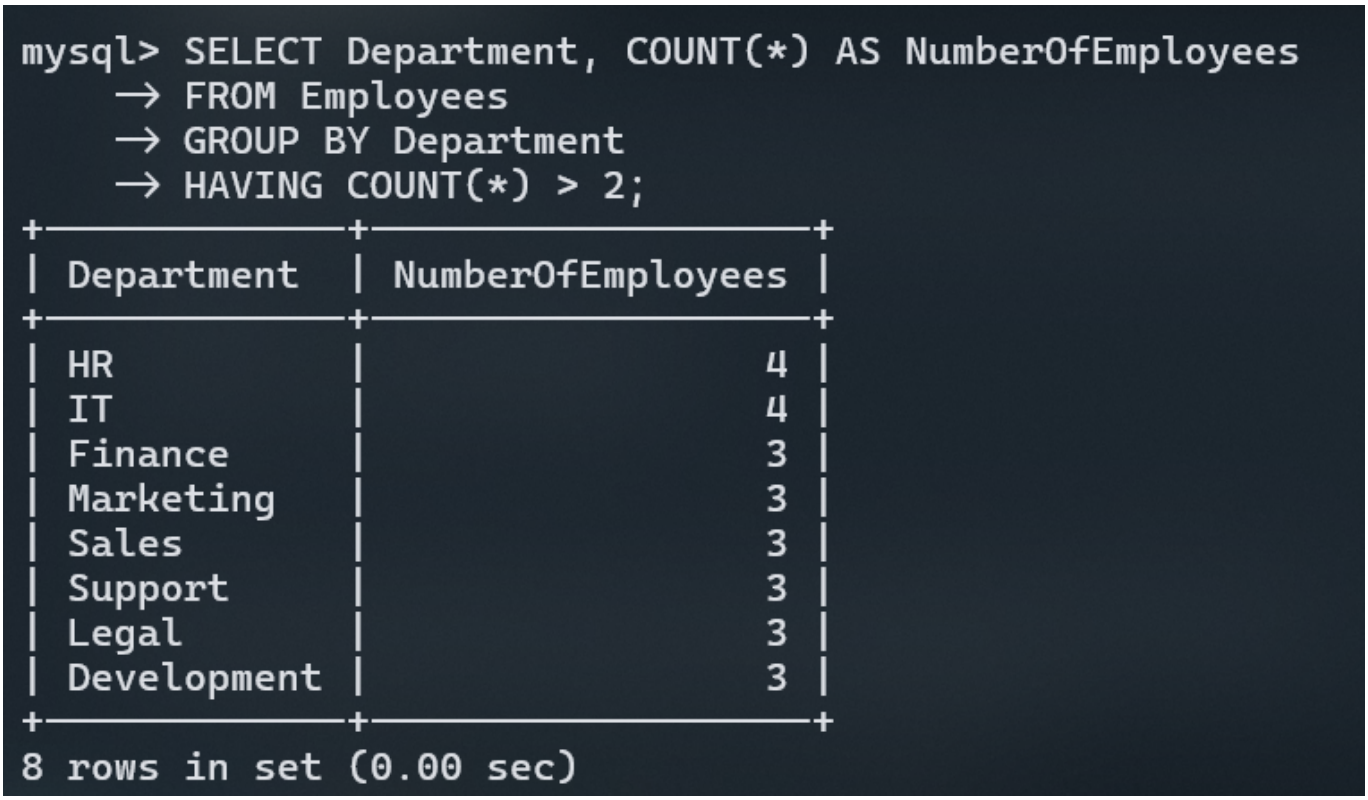
We can use the `HAVING` clause to list departments that have more than 2 employees.

```sql
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 2;
```

Explanation:

- **HAVING COUNT(*) > 2:** Filters departments where the count of employees is greater than 2.

Result:



```
mysql> SELECT Department, COUNT(*) AS NumberOfEmployees
    -> FROM Employees
    -> GROUP BY Department
    -> HAVING COUNT(*) > 2;
+-------------+---------------------+
| Department  | NumberOfEmployees   |
+-------------+---------------------+
| HR          |                   4 |
| IT          |                   4 |
| Finance     |                   3 |
| Marketing   |                   3 |
| Sales       |                   3 |
| Support     |                   3 |
| Legal       |                   3 |
| Development |                   3 |
+-------------+---------------------+
8 rows in set (0.00 sec)
```

Example 2: Filtering Departments with Total Salary Above $150,000

We can identify departments with a total salary exceeding $150,000.

```sql
SELECT Department, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Department
HAVING SUM(Salary) > 150000.00;
```

Explanation:

- **HAVING SUM(Salary) > 150000.00:** Filters departments where the total salary is greater than $150,000.

Result:

```
mysql> SELECT Department, SUM(Salary) AS TotalSalary
    -> FROM Employees
    -> GROUP BY Department
    -> HAVING SUM(Salary) > 150000.00;
+-------------+-------------+
| Department  | TotalSalary |
+-------------+-------------+
| HR          |   210000.00 |
| IT          |   250000.00 |
| Finance     |   205000.00 |
| Marketing   |   160000.00 |
| Legal       |   219000.00 |
| Development |   198000.00 |
+-------------+-------------+
6 rows in set (0.00 sec)
```

Example 3: Filtering Departments with Average Salary Less Than $55,000

To focus on departments where the average salary is below $55,000, use the following query:

```sql
SELECT Department, AVG(Salary) AS AverageSalary
FROM Employees
GROUP BY Department
HAVING AVG(Salary) < 55000.00;
```

Explanation:

- **HAVING AVG(Salary) < 55000.00:** Filters departments where the average salary is less than $55,000.

Result:

```
mysql> SELECT Department, AVG(Salary) AS AverageSalary
    -> FROM Employees
    -> GROUP BY Department
    -> HAVING AVG(Salary) < 55000.00;
+-------------+-----------------+
| Department  | AverageSalary   |
+-------------+-----------------+
| HR          |  52500.000000   |
| Marketing   |  53333.333333   |
| Sales       |  48000.000000   |
| Support     |  50000.000000   |
+-------------+-----------------+
4 rows in set (0.00 sec)
```

Example 4: Filtering Departments with Employees Who Joined After 2020

To list departments that have employees who joined after 2020 and have more than one employee, you can combine conditions using `HAVING`.

```sql
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
WHERE YEAR(JoinDate) > 2020
GROUP BY Department
HAVING COUNT(*) > 1;
```

Explanation:

- **WHERE YEAR(JoinDate) > 2020:** Filters employees who joined after 2020.
- **HAVING COUNT(*) > 1:** Filters departments with more than one such employee.

Result:

```
mysql> SELECT Department, COUNT(*) AS NumberOfEmployees
    -> FROM Employees
    -> WHERE YEAR(JoinDate) > 2020
    -> GROUP BY Department
    -> HAVING COUNT(*) > 1;
+-------------+---------------------+
| Department  | NumberOfEmployees   |
+-------------+---------------------+
| Finance     |                   2 |
| IT          |                   2 |
| Marketing   |                   3 |
| Support     |                   3 |
| Legal       |                   3 |
| Operations  |                   2 |
| Development |                   3 |
| Sales       |                   2 |
| Research    |                   2 |
+-------------+---------------------+
9 rows in set (0.00 sec)
```

Conclusion

Understanding the `GROUP BY` and `HAVING` clauses is essential for any SQL practitioner, as they allow you to efficiently aggregate and filter data. These clauses are powerful tools for generating meaningful insights from your data, whether for business analytics, financial reporting, or operational efficiency.