HAVING Clause

The HAVING clause in SQL is used to filter groups of rows returned by a GROUP BY clause. It's similar to the WHERE clause, but while the WHERE clause filters individual rows before they are grouped, the HAVING clause filters groups of rows after they have been grouped.

```
CREATE TABLE orders (
   order_id INT PRIMARY KEY AUTO_INCREMENT,
   customer_id INT,
   order_date DATE,
   total_amount DECIMAL(10, 2)
);
```

```
INSERT INTO orders (customer_id, order_date, total_amount) VALUES
(1, '2023-01-01', 500.00),
(1, '2023-01-15', 800.00),
(1, '2023-02-10', 1200.00),
(2, '2023-01-05', 700.00),
(2, '2023-02-20', 1500.00),
(2, '2023-03-01', 900.00),
(3, '2023-01-20', 1100.00),
(3, '2023-02-15', 1300.00),
(3, '2023-03-10', 2000.00),
(4, '2023-01-01', 500.00),
(4, '2023-01-15', 800.00),
(4, '2023-02-10', 1200.00),
(4, '2023-03-01', 900.00);
```

```
SELECT customer_id, COUNT(*) AS num_orders, SUM(total_amount) AS total_spent
FROM orders
GROUP BY customer_id
HAVING COUNT(*) > 3 AND SUM(total_amount) > 1000;
```

```
SELECT customer_id, COUNT(*) AS num_orders, SUM(total_amount) AS total_spent
FROM orders
WHERE YEAR(order_date) = 2023 AND MONTH(order_date) = 1
GROUP BY customer_id
HAVING SUM(total_amount) > 1000;
```

STORED Routine

A stored routine in SQL is a set of SQL statements that are stored on the database server and can be invoked repeatedly by client applications or other database objects. Stored routines include stored procedures and stored functions.

It have two types

1. Stored Procedure: A stored procedure is a precompiled collection of one or more SQL statements that can be executed as a single unit. It can accept input parameters, perform operations such as data manipulation or transaction control, and return output parameters or result sets. Stored procedures are typically used for complex logic or tasks that need to be executed frequently or reused across multiple parts of an application.

2. Stored Function: A stored function is similar to a stored procedure but differs in that it always returns a value. It accepts input parameters, performs calculations or data operations, and then returns a single value or a result set. Stored functions are often used to encapsulate business logic or computations that need to be reused in queries or other database objects.

Stored routines provide several benefits, including:

- Code reusability: Stored routines can be called from multiple parts of an application or other database objects, reducing the need to duplicate code.

- Improved performance: Stored routines are precompiled and stored on the server, which can improve execution speed compared to executing individual SQL statements.

- **Enhanced security: By encapsulating logic in stored routines, you can control access to sensitive data and operations, reducing the risk of unauthorized access or data manipulation.**

```
CREATE TABLE EmployeeDetails (
   employee_id INT PRIMARY KEY,
   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   email VARCHAR(100),
   phone_number VARCHAR(20),
   hire_date DATE,
   job_title VARCHAR(100),
   salary DECIMAL(10, 2)
);

INSERT INTO EmployeeDetails (employee_id, first_name, last_name, email, phone_number,
hire_date, job_title, salary)
VALUES
   (1, 'John', 'Doe', 'john.doe@example.com', '123-456-7890', '2023-01-15', 'Software
Engineer', 60000.00),
   (2, 'Jane', 'Smith', 'jane.smith@example.com', '987-654-3210', '2023-02-20', 'Data
Analyst', 55000.00),
   (3, 'Michael', 'Johnson', 'michael.johnson@example.com', '555-123-4567', '2023-03-10',
'Project Manager', 75000.00),
(4, 'Emily', 'Davis', 'emily.davis@example.com', '555-987-6543', '2023-04-05', 'Sales
Associate', 48000.00),
   (5, 'David', 'Brown', 'david.brown@example.com', '333-555-8888', '2023-05-20',
'Marketing Specialist', 60000.00),
   (6, 'Jennifer', 'Clark', 'jennifer.clark@example.com', '777-222-1111', '2023-06-15', 'Human
Resources Manager', 70000.00),
   (7, 'Daniel', 'Martinez', 'daniel.martinez@example.com', '111-444-7777', '2023-07-10',
'Financial Analyst', 65000.00),
   (8, 'Sarah', 'Taylor', 'sarah.taylor@example.com', '888-999-0000', '2023-08-25', 'Customer
Service Representative', 45000.00),
   (9, 'Robert', 'Wilson', 'robert.wilson@example.com', '555-111-2222', '2023-09-10',
'Software Engineer', 62000.00),
   (10, 'Laura', 'Anderson', 'laura.anderson@example.com', '111-222-3333', '2023-10-15',
'Software Engineer', 58000.00),
   (11, 'Thomas', 'Harris', 'thomas.harris@example.com', '333-444-5555', '2023-11-20',
'Software Engineer', 64000.00),
(12, 'Jessica', 'Miller', 'jessica.miller@example.com', '555-666-7777', '2023-12-10', 'Sales
Associate', 49000.00),
   (13, 'Ryan', 'Wilson', 'ryan.wilson@example.com', '777-888-9999', '2023-12-15', 'Sales
Associate', 50000.00),
```

```
   (14, 'Samantha', 'Thompson', 'samantha.thompson@example.com', '999-000-1111',
'2024-01-20', 'Sales Associate', 48000.00),
(15, 'Sophia', 'Jones', 'sophia.jones@example.com', '555-222-3333', '2024-02-05', 'Data
Analyst', 54000.00),
   (16, 'Ethan', 'White', 'ethan.white@example.com', '777-333-4444', '2024-03-10', 'Data
Analyst', 52000.00),
(18, 'Aiden', 'Anderson', 'aiden.anderson@example.com', '555-333-4444', '2024-05-20',
'Financial Analyst', 67000.00);

DELIMITER $$
CREATE PROCEDURE emp_info()
    BEGIN
       SELECT * FROM EmployeeDetails ORDER BY salary;
    END$$
DELIMITER ;
```

Now we will call the procedure

```
CALL database_name.procedure_name();
CALL college.emp_info();
Or
CALL emp_info();
```

Argument Passing in STORED PROCEDURE

```
DELIMITER $$
CREATE PROCEDURE get_empId(IN p_fname VARCHAR(50))
      BEGIN
      SELECT employee_id FROM EmployeeDetails
      WHERE first_name=p_fname;
END$$
DELIMITER ;
CALL get_empId('Daniel');
```

Return some value using a variable in STORED PROCEDURE

```
DELIMITER $$
mysql> CREATE PROCEDURE get_sum_through_fname(IN p_fname VARCHAR(100), OUT
p_sum DECIMAL(10,2))
   BEGIN
      SELECT SUM(salary) INTO p_sum FROM EmployeeDetails WHERE first_name=p_fname;
   END$$
DELIMITER ;
CALL get_sum_through_fname("Sarah", @p_sum);
SELECT @p_sum;
```

User Defined Functions

```
DELIMITER $$
CREATE FUNCTION emp_name_sal_max() RETURNS VARCHAR(50)
    DETERMINISTIC NO SQL READS SQL DATA
    BEGIN
    DECLARE v_max DECIMAL(10, 2);
    DECLARE v_emp_name VARCHAR(50);
    SELECT MAX(salary) INTO v_max FROM EmployeeDetails;
    SELECT first_name INTO v_emp_name FROM EmployeeDetails WHERE salary = v_max;
    RETURN v_emp_name;
    END$$

SELECT emp_name_sal_max();
```

Window Functions

It is also known as analytic functions allows us to perform calculations across a set of rows related to the current row.
It is defined by an OVER() clause.

```
SELECT
    SUM(salary) OVER() AS sum_salary
FROM EmployeeDetails;
```

```
+------------+
| sum_salary |
+------------+
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
|  982000.00 |
+------------+
```

```
SELECT
    employee_id,
    first_name,
     SUM(salary) OVER() AS sum_salary
 FROM EmployeeDetails;
```

employee_id	first_name	sum_salary
1	John	982000.00
2	Jane	982000.00
3	Michael	982000.00
4	Emily	982000.00
5	David	982000.00
6	Jennifer	982000.00
7	Daniel	982000.00
8	Sarah	982000.00
9	Robert	982000.00
10	Laura	982000.00
11	Thomas	982000.00
12	Jessica	982000.00
13	Ryan	982000.00
14	Samantha	982000.00
15	Sophia	982000.00
16	Ethan	982000.00
18	Aiden	982000.00

```
SELECT
    employee_id,
    first_name,
    salary,
     SUM(salary) OVER(ORDER BY employee_id) AS sum_salary
 FROM EmployeeDetails;
```

employee_id	first_name	salary	sum_salary
1	John	60000.00	60000.00
2	Jane	55000.00	115000.00
3	Michael	75000.00	190000.00
4	Emily	48000.00	238000.00
5	David	60000.00	298000.00
6	Jennifer	70000.00	368000.00
7	Daniel	65000.00	433000.00
8	Sarah	45000.00	478000.00
9	Robert	62000.00	540000.00
10	Laura	58000.00	598000.00
11	Thomas	64000.00	662000.00
12	Jessica	49000.00	711000.00
13	Ryan	50000.00	761000.00
14	Samantha	48000.00	809000.00
15	Sophia	54000.00	863000.00
16	Ethan	52000.00	915000.00
18	Aiden	67000.00	982000.00

```
SELECT
    employee_id,
    first_name,
    salary,
    job_title,
      SUM(salary) OVER(PARTITION BY job_title) AS sum_salary
 FROM EmployeeDetails;
```

employee_id	first_name	salary	job_title	sum_salary
8	Sarah	45000.00	Customer Service Representative	45000.00
2	Jane	55000.00	Data Analyst	161000.00
15	Sophia	54000.00	Data Analyst	161000.00
16	Ethan	52000.00	Data Analyst	161000.00
7	Daniel	65000.00	Financial Analyst	132000.00
18	Aiden	67000.00	Financial Analyst	132000.00
6	Jennifer	70000.00	Human Resources Manager	70000.00
5	David	60000.00	Marketing Specialist	60000.00
3	Michael	75000.00	Project Manager	75000.00
4	Emily	48000.00	Sales Associate	195000.00
12	Jessica	49000.00	Sales Associate	195000.00
13	Ryan	50000.00	Sales Associate	195000.00
14	Samantha	48000.00	Sales Associate	195000.00
1	John	60000.00	Software Engineer	244000.00
9	Robert	62000.00	Software Engineer	244000.00
10	Laura	58000.00	Software Engineer	244000.00
11	Thomas	64000.00	Software Engineer	244000.00

```
SELECT
    employee_id,
    first_name,
    salary,
    job_title,
      MAX(salary) OVER(PARTITION BY job_title) AS max_salary
 FROM EmployeeDetails;
```

employee_id	first_name	salary	job_title	max_salary
8	Sarah	45000.00	Customer Service Representative	45000.00
2	Jane	55000.00	Data Analyst	55000.00
15	Sophia	54000.00	Data Analyst	55000.00
16	Ethan	52000.00	Data Analyst	55000.00
7	Daniel	65000.00	Financial Analyst	67000.00
18	Aiden	67000.00	Financial Analyst	67000.00
6	Jennifer	70000.00	Human Resources Manager	70000.00
5	David	60000.00	Marketing Specialist	60000.00
3	Michael	75000.00	Project Manager	75000.00
4	Emily	48000.00	Sales Associate	50000.00
12	Jessica	49000.00	Sales Associate	50000.00
13	Ryan	50000.00	Sales Associate	50000.00
14	Samantha	48000.00	Sales Associate	50000.00
1	John	60000.00	Software Engineer	64000.00
9	Robert	62000.00	Software Engineer	64000.00
10	Laura	58000.00	Software Engineer	64000.00
11	Thomas	64000.00	Software Engineer	64000.00

ROW NUMBER()

The row number function in SQL is used to assign a unique sequential integer to each row in the result set. It is particularly useful for assigning a unique identifier to each row for further processing or analysis.

```
SELECT
    ROW_NUMBER() OVER() AS row_no,
    employee_id,
    salary,
    job_title
FROM EmployeeDetails;
```

row_no	employee_id	salary	job_title
1	1	60000.00	Software Engineer
2	2	55000.00	Data Analyst
3	3	75000.00	Project Manager
4	4	48000.00	Sales Associate
5	5	60000.00	Marketing Specialist
6	6	70000.00	Human Resources Manager
7	7	65000.00	Financial Analyst
8	8	45000.00	Customer Service Representative
9	9	62000.00	Software Engineer
10	10	58000.00	Software Engineer
11	11	64000.00	Software Engineer
12	12	49000.00	Sales Associate
13	13	50000.00	Sales Associate
14	14	48000.00	Sales Associate
15	15	54000.00	Data Analyst
16	16	52000.00	Data Analyst
17	18	67000.00	Financial Analyst

```
SELECT
    ROW_NUMBER() OVER(ORDER BY salary) AS row_no,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

row_no	employee_id	job_title	salary
1	8	Customer Service Representative	45000.00
2	4	Sales Associate	48000.00
3	14	Sales Associate	48000.00
4	12	Sales Associate	49000.00
5	13	Sales Associate	50000.00
6	16	Data Analyst	52000.00
7	15	Data Analyst	54000.00
8	2	Data Analyst	55000.00
9	10	Software Engineer	58000.00
10	1	Software Engineer	60000.00
11	5	Marketing Specialist	60000.00
12	9	Software Engineer	62000.00
13	11	Software Engineer	64000.00
14	7	Financial Analyst	65000.00
15	18	Financial Analyst	67000.00
16	6	Human Resources Manager	70000.00
17	3	Project Manager	75000.00

```
SELECT
    ROW_NUMBER() OVER(PARTITION BY job_title) AS row_no,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

row_no	employee_id	job_title	salary
1	8	Customer Service Representative	45000.00
1	2	Data Analyst	55000.00
2	15	Data Analyst	54000.00
3	16	Data Analyst	52000.00
1	7	Financial Analyst	65000.00
2	18	Financial Analyst	67000.00
1	6	Human Resources Manager	70000.00
1	5	Marketing Specialist	60000.00
1	3	Project Manager	75000.00
1	4	Sales Associate	48000.00
2	12	Sales Associate	49000.00
3	13	Sales Associate	50000.00
4	14	Sales Associate	48000.00
1	1	Software Engineer	60000.00
2	9	Software Engineer	62000.00
3	10	Software Engineer	58000.00
4	11	Software Engineer	64000.00

RANK()

The RANK() function in SQL is used to assign a rank to each row within the result set based on the values of specified columns. It is commonly used in scenarios where you want to determine the ranking of rows based on certain criteria.

```
SELECT
    RANK() OVER() AS rank_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

rank_salary	employee_id	job_title	salary
1	1	Software Engineer	60000.00
1	2	Data Analyst	55000.00
1	3	Project Manager	75000.00
1	4	Sales Associate	48000.00
1	5	Marketing Specialist	60000.00
1	6	Human Resources Manager	70000.00
1	7	Financial Analyst	65000.00
1	8	Customer Service Representative	45000.00
1	9	Software Engineer	62000.00
1	10	Software Engineer	58000.00
1	11	Software Engineer	64000.00
1	12	Sales Associate	49000.00
1	13	Sales Associate	50000.00
1	14	Sales Associate	48000.00
1	15	Data Analyst	54000.00
1	16	Data Analyst	52000.00
1	18	Financial Analyst	67000.00

```
SELECT
    RANK() OVER(ORDER BY salary) AS rank_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

rank_salary	employee_id	job_title	salary
1	8	Customer Service Representative	45000.00
2	4	Sales Associate	48000.00
2	14	Sales Associate	48000.00
4	12	Sales Associate	49000.00
5	13	Sales Associate	50000.00
6	16	Data Analyst	52000.00
7	15	Data Analyst	54000.00
8	2	Data Analyst	55000.00
9	10	Software Engineer	58000.00
10	1	Software Engineer	60000.00
10	5	Marketing Specialist	60000.00
12	9	Software Engineer	62000.00
13	11	Software Engineer	64000.00
14	7	Financial Analyst	65000.00
15	18	Financial Analyst	67000.00
16	6	Human Resources Manager	70000.00
17	3	Project Manager	75000.00

```
SELECT
    RANK() OVER(ORDER BY salary DESC) AS rank_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

rank_salary	employee_id	job_title	salary
1	3	Project Manager	75000.00
2	6	Human Resources Manager	70000.00
3	18	Financial Analyst	67000.00
4	7	Financial Analyst	65000.00
5	11	Software Engineer	64000.00
6	9	Software Engineer	62000.00
7	1	Software Engineer	60000.00
7	5	Marketing Specialist	60000.00
9	10	Software Engineer	58000.00
10	2	Data Analyst	55000.00
11	15	Data Analyst	54000.00
12	16	Data Analyst	52000.00
13	13	Sales Associate	50000.00
14	12	Sales Associate	49000.00
15	4	Sales Associate	48000.00
15	14	Sales Associate	48000.00
17	8	Customer Service Representative	45000.00

DENSE_RANK()

The DENSE_RANK() function in SQL is similar to the RANK() function, but it handles tied ranks differently. While the RANK() function leaves gaps in the ranking sequence when there are ties, the DENSE_RANK() function does not.

```
SELECT
    DENSE_RANK() OVER(ORDER BY salary) AS rank_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

rank_salary	employee_id	job_title	salary
1	8	Customer Service Representative	45000.00
2	4	Sales Associate	48000.00
2	14	Sales Associate	48000.00
3	12	Sales Associate	49000.00
4	13	Sales Associate	50000.00
5	16	Data Analyst	52000.00
6	15	Data Analyst	54000.00
7	2	Data Analyst	55000.00
8	10	Software Engineer	58000.00
9	1	Software Engineer	60000.00
9	5	Marketing Specialist	60000.00
10	9	Software Engineer	62000.00
11	11	Software Engineer	64000.00
12	7	Financial Analyst	65000.00
13	18	Financial Analyst	67000.00
14	6	Human Resources Manager	70000.00
15	3	Project Manager	75000.00

LAG()

The LAG() function in SQL is a window function that allows you to access data from a previous row within the result set. It is commonly used to retrieve the value of a column from the previous row, based on a specified order.

```
SELECT
    LAG(salary) OVER() AS lag_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

lag_salary	employee_id	job_title	salary
NULL	1	Software Engineer	60000.00
60000.00	2	Data Analyst	55000.00
55000.00	3	Project Manager	75000.00
75000.00	4	Sales Associate	48000.00
48000.00	5	Marketing Specialist	60000.00
60000.00	6	Human Resources Manager	70000.00
70000.00	7	Financial Analyst	65000.00
65000.00	8	Customer Service Representative	45000.00
45000.00	9	Software Engineer	62000.00
62000.00	10	Software Engineer	58000.00
58000.00	11	Software Engineer	64000.00
64000.00	12	Sales Associate	49000.00
49000.00	13	Sales Associate	50000.00
50000.00	14	Sales Associate	48000.00
48000.00	15	Data Analyst	54000.00
54000.00	16	Data Analyst	52000.00
52000.00	18	Financial Analyst	67000.00

LEAD()

The LEAD() function in SQL is a window function that allows you to access data from a subsequent row within the result set. It is the counterpart to the LAG() function, which accesses data from a previous row.

```
SELECT
    LEAD(salary) OVER() AS lag_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

lag_salary	employee_id	job_title	salary
55000.00	1	Software Engineer	60000.00
75000.00	2	Data Analyst	55000.00
48000.00	3	Project Manager	75000.00
60000.00	4	Sales Associate	48000.00
70000.00	5	Marketing Specialist	60000.00
65000.00	6	Human Resources Manager	70000.00
45000.00	7	Financial Analyst	65000.00
62000.00	8	Customer Service Representative	45000.00
58000.00	9	Software Engineer	62000.00
64000.00	10	Software Engineer	58000.00
49000.00	11	Software Engineer	64000.00
50000.00	12	Sales Associate	49000.00
48000.00	13	Sales Associate	50000.00
54000.00	14	Sales Associate	48000.00
52000.00	15	Data Analyst	54000.00
67000.00	16	Data Analyst	52000.00
NULL	18	Financial Analyst	67000.00

```
SELECT
    Salary - LAG(salary) OVER(ORDER BY salary DESC) AS diff_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

diff_salary	employee_id	job_title	salary
NULL	3	Project Manager	75000.00
-5000.00	6	Human Resources Manager	70000.00
-3000.00	18	Financial Analyst	67000.00
-2000.00	7	Financial Analyst	65000.00
-1000.00	11	Software Engineer	64000.00
-2000.00	9	Software Engineer	62000.00
-2000.00	1	Software Engineer	60000.00
0.00	5	Marketing Specialist	60000.00
-2000.00	10	Software Engineer	58000.00
-3000.00	2	Data Analyst	55000.00
-1000.00	15	Data Analyst	54000.00
-2000.00	16	Data Analyst	52000.00
-2000.00	13	Sales Associate	50000.00
-1000.00	12	Sales Associate	49000.00
-1000.00	4	Sales Associate	48000.00
0.00	14	Sales Associate	48000.00
-3000.00	8	Customer Service Representative	45000.00

```
SELECT
    Salary - LEAD(salary) OVER(ORDER BY salary DESC) AS diff_salary,
    employee_id,
    job_title,
    salary
FROM EmployeeDetails;
```

diff_salary	employee_id	job_title	salary
5000.00	3	Project Manager	75000.00
3000.00	6	Human Resources Manager	70000.00
2000.00	18	Financial Analyst	67000.00
1000.00	7	Financial Analyst	65000.00
2000.00	11	Software Engineer	64000.00
2000.00	9	Software Engineer	62000.00
0.00	1	Software Engineer	60000.00
2000.00	5	Marketing Specialist	60000.00
3000.00	10	Software Engineer	58000.00
1000.00	2	Data Analyst	55000.00
2000.00	15	Data Analyst	54000.00
2000.00	16	Data Analyst	52000.00
1000.00	13	Sales Associate	50000.00
1000.00	12	Sales Associate	49000.00
0.00	4	Sales Associate	48000.00
3000.00	14	Sales Associate	48000.00
NULL	8	Customer Service Representative	45000.00