POWERED BY:
BEGINNERSBLOG.ORG

Complete Guide to GROUP BY and HAVING in SQL

By SHAILESH SHAKYA @BEGINNERSBLOG.ORG

Understanding GROUP BY in SQL

The GROUP BY clause in SQL is used to arrange identical data into groups. It is often used with aggregate functions such as SUM(), COUNT(), AVG(), MAX(), and MIN() to perform calculations on each group of data.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name;
```

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

Key Points:

- GROUP BY groups the result set by one or more columns.
- It must be used when you have aggregate functions in your query.
- The columns in the SELECT statement must either be included in GROUP BY or used within an aggregate function.
- The result set contains one row per group.

Swipe to Next Slide

Understanding HAVING in SQL

The HAVING clause is used to filter records that work on aggregated data. Unlike WHERE, which filters records before aggregation, HAVING filters records after aggregation.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

Key Points:

- HAVING is used only with GROUP BY.
- It filters grouped records based on aggregate function results.
- It works like WHERE but on grouped data.

When to Use GROUP BY in SQL Queries?

1. You need to aggregate data (e.g., finding totals, averages, min, max, or count per category).
2. You need to summarize large data sets into smaller meaningful groups.
3. You need to analyze patterns or trends across different categories.
4. You need to apply HAVING for filtering after aggregation.

Swipe to Next Slide

Examples of GROUP BY and HAVING with Aggregate Functions

1. Counting Customers Per Country

```sql
SELECT country, COUNT(customer_id) AS total_customers
FROM customers
GROUP BY country;
```

2. Finding the Average Order Value Per Customer

```sql
SELECT customer_id, AVG(order_amount) AS average_order_value
FROM orders
GROUP BY customer_id;
```

3. Identifying the Maximum Salary Per Department

```sql
SELECT department, MAX(salary) AS max_salary
FROM employees
GROUP BY department;
```

4. Summing Up Total Sales Per Product

```sql
SELECT product_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY product_id;
```

Swipe to Next Slide

By SHAILESH SHAKYA

5. Filtering Departments with an Average Salary Above 50,000

```sql
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

6. Finding Products with More Than 100 Orders

```sql
SELECT product_id, COUNT(order_id) AS total_orders
FROM orders
GROUP BY product_id
HAVING COUNT(order_id) > 100;
```

7. Listing Cities Where Total Sales Exceed $1M

```sql
SELECT city, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY city
HAVING SUM(sales_amount) > 1000000;
```

8. Finding Customers Who Have Placed More Than 10 Orders

```sql
SELECT customer_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY customer_id
HAVING COUNT(order_id) > 10;
```

9. Identifying Stores with Minimum Sales Below $500

```sql
SELECT store_id, MIN(sales_amount) AS min_sales
FROM sales
GROUP BY store_id
HAVING MIN(sales_amount) < 500;
```

10. Getting the Average Age of Employees Per Job Title

```sql
SELECT job_title, AVG(age) AS average_age
FROM employees
GROUP BY job_title;
```

11. Finding the Total Revenue Per Year

```sql
SELECT YEAR(order_date) AS order_year, SUM(order_amount) AS tota
FROM orders
GROUP BY YEAR(order_date);
```

12. Identifying Customers Who Spent More Than $5000 in Total

```sql
SELECT customer_id, SUM(order_amount) AS total_spent
FROM orders
GROUP BY customer_id
HAVING SUM(order_amount) > 5000;
```

Swipe to Next Slide

13. Counting Employees Per Job Title in Each Department

```sql
SELECT department, job_title, COUNT(employee_id) AS employee_cou
FROM employees
GROUP BY department, job_title;
```

14. Identifying the Earliest and Latest Order Date Per Customer

```sql
SELECT customer_id, MIN(order_date) AS first_order, MAX(order_da
FROM orders
GROUP BY customer_id;
```

15. Finding The Store with The Highest Number of Orders

```sql
SELECT store_id, COUNT(order_id) AS order_count
FROM sales
GROUP BY store_id
ORDER BY order_count DESC
LIMIT 1;
```

16. Calculating the Monthly Sales Total for Each Product

```sql
SELECT product_id, MONTH(order_date) AS order_month, SUM(sales_a
FROM sales
GROUP BY product_id, MONTH(order_date);
```

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

17. Finding the Most Frequently Ordered Product

```sql
SELECT product_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY product_id
ORDER BY order_count DESC
LIMIT 1;
```

18. Identifying Employees with the Longest Tenure Per Department

```sql
SELECT department, employee_id, MAX(years_of_service) AS max_ter
FROM employees
GROUP BY department, employee_id;
```

19. Finding Average Ratings Per Product with More Than 50 Reviews

```sql
SELECT product_id, AVG(rating) AS avg_rating
FROM reviews
GROUP BY product_id
HAVING COUNT(review_id) > 50;
```

20. Summing Sales Per Product Per Quarter

```sql
SELECT product_id, QUARTER(order_date) AS order_quarter, SUM(sa
FROM sales
GROUP BY product_id, QUARTER(order_date);
```

Key Takeaways

- GROUP BY is used to group rows with common values.
- It is always used with aggregate functions to compute statistics per group.
- HAVING filters the results after aggregation.
- These clauses are essential for analytical SQL queries, reporting, and business intelligence.

Swipe to Next Slide

🔥 Special Use Cases & Advanced Techniques

1️⃣ Using GROUP BY with Multiple Columns
You can group by multiple columns to create more granular groupings.

```
SELECT department, job_title,
COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department, job_title;
```

📌 Tip: The more columns you add to GROUP BY, the more specific the grouping.

Swipe to Next Slide

2️⃣ Using GROUP BY with DATE/TIME Functions

If you're working with sales data, you often need to group by year, month, week, or day.

```
SELECT YEAR(order_date) AS order_year,
MONTH(order_date) AS order_month,
SUM(order_amount) AS total_sales
FROM orders
GROUP BY YEAR(order_date),
MONTH(order_date);
```

📌 Tip: Use YEAR(), MONTH(), DAY(), WEEK(), QUARTER() for better time-based aggregation.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

3 Using GROUP BY with CASE Statements

You can use CASE inside GROUP BY to categorize data dynamically.

```
SELECT
    CASE
        WHEN age < 18 THEN 'Minor'
        WHEN age BETWEEN 18 AND 60 THEN
'Adult'
        ELSE 'Senior'
    END AS age_group, COUNT(*) AS count
FROM customers
GROUP BY age_group;
```

📌 Tip: This is useful for grouping based on custom conditions.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

4 Filtering Without HAVING

If filtering before grouping, use WHERE. If filtering after aggregation, use HAVING.

```
SELECT department, COUNT(*) AS
employee_count
FROM employees
WHERE salary > 50000  – Filters before grouping
GROUP BY department
HAVING COUNT(*) > 5;  – Filters after
aggregation
```

📌 Tip: WHERE is for raw data, HAVING is for aggregated results.

Swipe to Next Slide

5 Using GROUP BY with DISTINCT

If you want unique values before applying GROUP BY, use DISTINCT.

```
SELECT department, COUNT(DISTINCT
job_title) AS unique_jobs
FROM employees
GROUP BY department;
```

📌 Tip: This ensures that only unique job titles are counted per department.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

6 Using WINDOW FUNCTIONS Instead of GROUP BY

Sometimes, GROUP BY is not the best option. You can use window functions to avoid losing row details.

```
SELECT employee_id, department, salary,
    AVG(salary) OVER (PARTITION BY
department) AS avg_salary_per_dept
FROM employees;
```

📌 Tip: Window functions (OVER(PARTITION BY)) allow aggregate calculations without collapsing rows.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

7 Using ROLLUP for Subtotals

If you need subtotal calculations, use ROLLUP.

```
SELECT department, job_title, COUNT(*) AS
count
FROM employees
GROUP BY department, job_title WITH
ROLLUP;
```

📌 Tip: This gives you an extra row for each subtotal per department.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

8 Using CUBE for Multi-Dimensional Analysis

CUBE calculates all possible groupings.

```
SELECT department, job_title,
COUNT(*) AS count
FROM employees
GROUP BY CUBE(department, job_title);
```

📌 Tip: This is useful for advanced reporting like pivot tables.

Swipe to Next Slide

9 Finding Duplicates Using GROUP BY

To find duplicate records in a table, use GROUP BY with HAVING COUNT(*) > 1.

```
SELECT email, COUNT(*) AS duplicate_count
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

📌 Tip: Use this technique to identify duplicate records in a database.

Swipe to Next Slide

🔟 Using GROUP BY with JSON Aggregation

For modern SQL databases, you can aggregate data into JSON format.

```
SELECT customer_id,
JSON_ARRAYAGG(order_id) AS order_list
FROM orders
GROUP BY customer_id;
```

📌 Tip: This is great for API responses or structured data storage.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

🔟 Using GROUP BY with JSON Aggregation

For modern SQL databases, you can aggregate data into JSON format.

```
SELECT customer_id,
JSON_ARRAYAGG(order_id) AS order_list
FROM orders
GROUP BY customer_id;
```

📌 Tip: This is great for API responses or structured data storage.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

🔥 Pro Tips for Mastering GROUP BY and HAVING

✅ Always include non-aggregated columns in GROUP BY

✅ Use Aliases for Readability

```
SELECT department, COUNT(employee_id) AS
total_employees
FROM employees
GROUP BY department
HAVING total_employees > 5;  – Error (Aliases
don’t work in HAVING)
```

❌ The alias total_employees cannot be used in HAVING. Instead, repeat the aggregate function:

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

✅ ORDER BY After GROUP BY

You can sort grouped data using ORDER BY.

```
SELECT department, COUNT(*) AS
employee_count
FROM employees
GROUP BY department
ORDER BY employee_count DESC;
```

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

✅ HAVING Without GROUP BY

If you use aggregate functions without GROUP BY, SQL considers all rows as one group.

```
SELECT COUNT(*) AS total_orders
FROM orders
HAVING COUNT(*) > 1000;
```

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

✅ Performance Optimization Tips

1. Use Indexing: Speed up GROUP BY queries by indexing grouped columns.
2. Filter with WHERE Before GROUP BY: Reduces the dataset before aggregation.
3. Avoid GROUP BY on High-Cardinality Columns: Grouping on unique columns (like id) creates large result sets.

Swipe to Next Slide

By SHAILESH SHAKYA

POWERED BY: BEGINNERSBLOG.ORG

I hope you have found this **ROADMAP** actionable to become a **BIG DATA ENGINEER/ARCHITECT**

Join **OpenAILearning** to get more educational stuff Similar to this you finished reading ↘ ↘

Telegram: **OpenAILearning**

WhatsApp: **OpenAILearning**

Thank You!

Swipe to Next Slide

By SHAILESH SHAKYA

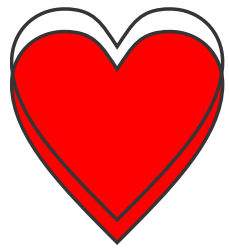
POWERED BY: BEGINNERSBLOG.ORG

Created by Shailesh Shakya

@BEGINNERSBLOG.ORG

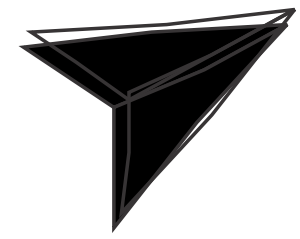
Did you find this post helpful?

Please...

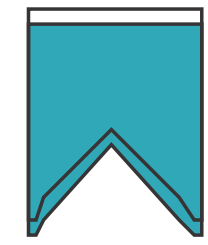


LIKE

COMMENT



REPOST



SAVE