**SQL GROUP BY**

The GROUP BY clause in SQL is a powerful feature used to group rows with the same values into summary rows. It is often used in combination with aggregate functions (such as COUNT, SUM, AVG, MAX, MIN) to perform calculations on groups of data rather than individual rows. Here's a detailed explanation of how the GROUP BY clause works:

1. **Grouping Rows**: When you use the GROUP BY clause, you're telling the database to group together rows that have the same values in specified columns. Each unique combination of values in the specified columns forms a group.
2. **Aggregate Functions**: Once the rows are grouped, you can perform aggregate functions on each group to calculate summary statistics. These functions operate on the values within each group and return a single result for the group.
3. **Syntax**: The basic syntax of a GROUP BY query is as follows:

SELECT column1, column2, ..., aggregate\_function(column\_name)

FROM table\_name

GROUP BY column1, column2, ...;

1. **Columns to Group By**: In the GROUP BY clause, you specify the columns by which you want to group the data. These columns can be any columns in the table or expressions based on columns.
2. **Aggregate Functions**: Inside the SELECT statement, you can use aggregate functions to perform calculations on each group. These functions operate on the values in the grouped columns and produce a single result for each group.
3. **Result Set**: The result set of a GROUP BY query consists of one row for each group. The columns in the result set can include the grouped columns along with the results of aggregate functions applied to each group.
4. **Order of Execution**: In SQL, the GROUP BY clause is typically executed after the WHERE clause and before the ORDER BY clause. This means that rows are first filtered based on the conditions in the WHERE clause, then grouped according to the columns specified in the GROUP BY clause, and finally, the result set is ordered based on the criteria specified in the ORDER BY clause.
5. **Filtering Groups**: You can use the HAVING clause to filter groups based on aggregate values. The HAVING clause is similar to the WHERE clause but is used specifically for filtering groups rather than individual rows.
6. **Null Values**: When using GROUP BY, NULL values are treated as a single group. This means that if a column contains NULL values, all of those rows will be grouped together.
7. **Performance Considerations**: Grouping large datasets can impact performance, especially when using aggregate functions on many rows. It's essential to use indexes, proper database design, and efficient queries to optimize performance when using GROUP BY.

Overall, the GROUP BY clause is essential for performing analysis and generating summary statistics on grouped data in SQL. It allows you to organize and analyze data effectively, making it a fundamental tool in database querying and reporting.

**SQL Aggregate Functions**

An SQL aggregate function calculates on a set of values and returns a single value. For example, the average function (AVG) takes a list of values and returns the average.

Because an aggregate function operates on a set of values, it is often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause divides the result set into groups of values and the aggregate function returns a single value for each group.

The following are the commonly used SQL aggregate functions:

* AVG() – returns the average of a set.
* COUNT() – returns the number of items in a set.
* MAX() – returns the maximum value in a set.
* MIN() – returns the minimum value in a set
* SUM() – returns the sum of all or distinct values in a set

1. Calculate the average grade for each major.

SELECT Major, AVG(Grade) AS AverageGrade FROM Student GROUP BY Major;

1. Count the number of students in each major.

SELECT Major, COUNT(\*) AS NumberOfStudents FROM Student GROUP BY Major;

1. Find the maximum age among students in each major.

SELECT Major, MAX(Age) AS MaxAge FROM Student GROUP BY Major;

1. Calculate the total grade points for each major.

SELECT Major, SUM(Grade) AS TotalGradePoints FROM Student GROUP BY Major;

1. Find the number of students of each age.

SELECT Age, COUNT(\*) AS NumberOfStudents FROM Student GROUP BY Age;

1. Calculate the average age of students in each major.

SELECT Major, AVG(Age) AS AverageAge FROM Student GROUP BY Major;

1. Find the minimum grade among students in each major.

SELECT Major, MIN(Grade) AS MinGrade FROM Student GROUP BY Major;

1. Count the number of students for each combination of major and age.

SELECT Major, Age, COUNT(\*) AS NumberOfStudents FROM Student GROUP BY Major, Age;

1. Calculate the total number of students and average grade for each major and age combination.

SELECT Major, Age, COUNT(\*) AS NumberOfStudents, AVG(Grade) AS AverageGrade FROM Student GROUP BY Major, Age;

1. Find the major with the highest average grade.

SELECT TOP (1) Major, AVG(Grade) AS avg\_grade FROM Student GROUP BY Major ORDER BY avg\_grade DESC;

1. Calculate the sum of grades for each major and age combination.

SELECT Major, Age, SUM(Grade) AS TotalGrade FROM Student GROUP BY Major, Age;

1. Find the major(s) with the most number of students.

SELECT Major, COUNT(\*) AS NumberOfStudents FROM Student GROUP BY Major ORDER BY NumberOfStudents DESC;

1. Calculate the average age of students and the average grade for each major.

SELECT Major, AVG(Age) AS AverageAge, AVG(Grade) AS AverageGrade FROM Student GROUP BY Major;

1. Find the major(s) with the highest total grade points.

SELECT Major, SUM(Grade) AS TotalGradePoints FROM Student GROUP BY Major ORDER BY TotalGradePoints DESC;

1. Calculate the difference between the maximum and minimum age for each major.

SELECT Major, MAX(Age) - MIN(Age) AS AgeDifference FROM Student GROUP BY Major;

1. Find the major(s) with the highest minimum grade.

SELECT Major, MIN(Grade) AS MinGrade FROM Student GROUP BY Major ORDER BY MinGrade DESC;

1. Calculate the average grade for each major and order the results by average grade in descending order.

SELECT Major, AVG(Grade) AS AverageGrade FROM Student GROUP BY Major ORDER BY AverageGrade DESC;

1. Find the major(s) with the highest number of students aged 20.

SELECT Major, COUNT(\*) AS NumberOfStudents FROM Student WHERE Age = 20 GROUP BY Major ORDER BY NumberOfStudents DESC;

1. Calculate the difference between the maximum and minimum grade for each major.

SELECT Major, MAX(Grade) - MIN(Grade) AS GradeRange FROM Student GROUP BY Major;

**SQL HAVING**

In SQL, the HAVING clause is used in combination with the GROUP BY clause to filter the rows that are returned by a query based on aggregate values. While the WHERE clause is used to filter individual rows before any grouping or aggregation occurs, the HAVING clause allows you to filter groups of rows after they have been grouped using the GROUP BY clause and aggregate functions have been applied.

Here's a detailed explanation of the HAVING clause:

* 1. Usage: The HAVING clause is used in SQL queries to apply conditions to groups of rows defined by the GROUP BY clause. It is typically placed after the GROUP BY clause and before the ORDER BY clause in a SELECT statement.
  2. Syntax: The basic syntax of a query with a HAVING clause is similar to that of a query with a WHERE clause, but it operates on aggregated data rather than individual rows:

SELECT column1, column2, ..., aggregate\_function(column\_name)

FROM table\_name

GROUP BY column1, column2, ...

HAVING condition;

* 1. Filtering Groups: The condition specified in the HAVING clause is applied to each group of rows after they have been grouped using the GROUP BY clause. It allows you to filter groups based on aggregate values computed for each group.
  2. Aggregate Functions: Typically, the condition in the HAVING clause involves aggregate functions (such as COUNT, SUM, AVG, MAX, MIN) that operate on columns within each group. You can compare the result of aggregate functions to specific values or use logical operators to define more complex conditions.
  3. Difference from WHERE Clause: While the WHERE clause filters individual rows based on conditions before any grouping occurs, the HAVING clause filters groups of rows based on conditions after they have been grouped using the GROUP BY clause. Therefore, the HAVING clause can only be used in SELECT statements that also include a GROUP BY clause.

Example: Suppose you have a table of orders and you want to find the total sales amount for each customer, but you only want to include customers whose total sales amount exceeds a certain threshold (let's say $1000). You can use the HAVING clause to filter out the groups of customers who don't meet this condition after grouping by customer:

SELECT customer\_id, SUM(amount) AS total\_sales

FROM orders

GROUP BY customer\_id

HAVING SUM(amount) > 1000;

Overall, the HAVING clause is a valuable tool in SQL for filtering groups of rows based on aggregate values, allowing for more complex analysis and reporting on grouped data.

**Note:** add some records to STUDENT table:

INSERT INTO Student VALUES

(10,'Ethan', 'Jones', 21, 75, 'Mathematics'),

(11,'Olivia', 'Wilson', 20, 82, 'Physics'),

(12,'Liam', 'Taylor', 23, 90, 'Chemistry'),

(13,'Ava', 'Anderson', 22, 78, 'Biology'),

(14,'Noah', 'Brown', 21, 85, 'English'),

(15,'Sophia', 'Clark', 20, 91, 'Computer Science'),

(16,'William', 'Martinez', 23, 79, 'Mathematics'),

(17,'Isabella', 'Hernandez', 22, 84, 'Physics'),

(18,'James', 'Young', 21, 87, 'Chemistry'),

(19,'Emma', 'Smith', 22, 88, 'Computer Science');

**Examples:**

the "Student" table has the following columns: StudentID, FirstName, LastName, Age, Grade, Major.

1- Filtering based on average grade greater than 80:

SELECT Major, AVG(Grade) AS avg\_grade FROM Student GROUP BY Major HAVING AVG(Grade) > 80;

2- Filtering based on the number of students in each major being more than 1:

SELECT Major, COUNT(\*) AS num\_students FROM Student GROUP BY Major HAVING COUNT(\*) > 1;

3- Filtering based on total grade points being greater than 200:

SELECT Major, SUM(Grade) AS total\_grade FROM Student GROUP BY Major HAVING SUM(Grade) > 200;

4- Filtering based on the maximum age in each major being greater than 22:

SELECT Major, MAX(Age) AS max\_age FROM Student GROUP BY Major HAVING MAX(Age) > 22;

5- Filtering based on the minimum grade in each major being smaller than 80:

SELECT Major, MIN(Grade) AS min\_grade FROM Student GROUP BY Major HAVING MIN(Grade) < 80;

6- Filtering based on multiple aggregate conditions (average grade, being greater than 75 and max age being greater than 21):

SELECT Major, AVG(Grade) AS avg\_grade, MAX(Age) AS max\_age FROM Student GROUP BY Major HAVING AVG(Grade) > 75 AND MAX(Age) > 21;

1. Filtering based on the average age in each major being smaller than 25:

SELECT Major, AVG(Age) AS avg\_age FROM Student GROUP BY Major HAVING AVG(Age) < 25;

1. Filtering based on a combination of aggregate functions (average grade, being greater than 80 and number of students being greater than 2):

SELECT Major, COUNT(\*) AS num\_students, AVG(Grade) AS avg\_grade FROM Student GROUP BY Major HAVING COUNT(\*) > 2 AND AVG(Grade) > 80;

1. Filtering based on the difference between the maximum and minimum age being greater than 2:

SELECT Major, MAX(Age) - MIN(Age) AS age\_range FROM Student GROUP BY Major HAVING MAX(Age) - MIN(Age) > 2;

1. Filtering based on the difference between the maximum and minimum grade (being greater than 10):

SELECT Major, MAX(Grade) - MIN(Grade) AS grade\_range FROM Student

GROUP BY Major HAVING MAX(Grade) - MIN(Grade) > 10;