Group by clause

The GROUP BY clause in MySQL is used to group rows that have the same values in one or more columns. It is used in conjunction with aggregate functions such as SUM, AVG, MAX, MIN, COUNT, etc. to summarize data based on specific columns.

Syntax

```
SELECT column_name1, aggregate_function(column_name2)
FROM table_name
GROUP BY column_name1;
```

In this syntax, column_name1 is the column or columns that you want to group the data by, and column_name2 is the column that you want to apply the aggregate function to.

Example 1:

```
+----+-------+--------+
| id | name  | amount |
+----+-------+--------+
|  1 | John  |   1000 |
|  2 | Peter |    500 |
|  3 | John  |    750 |
|  4 | Mary  |    900 |
|  5 | John  |    600 |
+----+-------+--------+
```

If you want to know the total amount of sales for each person, you can use the GROUP BY clause as follows:

```
SELECT name, SUM(amount) as total_sales
FROM sales
GROUP BY name;
```

The result will be:

```
+-------+-------------+
| name  | total_sales |
+-------+-------------+
| John  |        2350 |
| Mary  |         900 |
| Peter |         500 |
+-------+-------------+
```

Here, the data is grouped by the name column, and the SUM function is used to calculate the total amount of sales for each person. The result shows the total sales for each person in the table.

Order by and Having clauses

Two commonly used clauses in conjunction with GROUP BY are ORDER BY and HAVING. ORDER BY is used to sort the result set based on one or more columns, either in ascending or descending order. HAVING is used to filter the results after grouping has been done, and is used to specify a condition on the result of an aggregate function.

Syntax

```
SELECT column_name1, aggregate_function(column_name2)
FROM table_name
GROUP BY column_name1
HAVING condition
ORDER BY column_name1 [ASC|DESC];
```

In this syntax, column_name1 is the column or columns that you want to group the data by, and column_name2 is the column that you want to apply the aggregate function to. condition is the condition that the aggregate function result must satisfy. ASC and DESC are used to specify the sorting order.

Example 2:

For example, let's say we have a table named orders with the following data:

```
+----+-------+--------+
| id | name  | amount |
+----+-------+--------+
|  1 | John  |   1000 |
|  2 | Peter |    500 |
|  3 | John  |    750 |
|  4 | Mary  |    900 |
|  5 | John  |    600 |
+----+-------+--------+
```

If we want to find the total amount of sales for each person, but only include people with a total sales amount greater than 1000 and sort the results in descending order based on the total sales amount, we could use the following query:

```
SELECT name, SUM(amount) as total_sales
FROM orders
GROUP BY name
HAVING total_sales > 1000
ORDER BY total_sales DESC;
```

The result would be:

```
+------+-------------+
| name | total_sales |
+------+-------------+
| John |        2350 |
+------+-------------+
```

Here, the data is grouped by the name column, and the SUM function is used to calculate the total amount of sales for each person. The HAVING clause filters out anyone with a total sales amount less than or equal to 1000. Finally, the ORDER BY clause sorts the results in descending order based on the total sales amount.

Aggregate functions

MySQL supports several aggregate functions that can be used to perform calculations on groups of rows in a table. Here are some commonly used aggregate functions in MySQL:

1. COUNT(): This function is used to count the number of rows that match a specified condition.
2. SUM(): This function is used to calculate the sum of a numeric column.
3. AVG(): This function is used to calculate the average of a numeric column.
4. MAcX(): This function is used to find the maximum value of a column.
5. MIN(): This function is used to find the minimum value of a column.

Example 3:

Find the sum of the amounts paid by the customer from payment with id = 155.
select sum(amount) from payment where customer_id=155;

Example 4:

Find the count of items of inventory at each store.
select store_id,count(inventory_id) from inventory group by store_id;

Example 5:

Find the average length of the films with rating = PG. (use film table)
select avg(length) from film where rating='PG';

Multi-column grouping and Rolls ups

Multi-column grouping in MySQL allows you to group data by multiple columns at the same time. This is useful when you want to analyze data that has multiple dimensions or attributes.
To use multi-column grouping, you simply list the columns you want to group by in the GROUP BY clause, separated by commas.

Example 6:

You have a table of sales data with columns for region, year, and month, you could group the data by region and year using the following query:

```
SELECT region, year, SUM(sales) as total_sales
FROM sales_data
GROUP BY region, year;
```

wThis query will group the data by both the region and year columns, and calculate the total sales for each combination of region and year.

Rollups

Multi-column grouping with rollups in MySQL allows you to group data by multiple columns and also include summary rows that provide subtotals for each level of grouping.

To use rollups, you add the WITH ROLLUP modifier to the GROUP BY clause.

Example 7:

You have a table of sales data with columns for region, year, and month, you could group the data by region and year with rollups using the following query:

```
SELECT region, year, SUM(sales) as total_sales
FROM sales_data
GROUP BY region, year WITH ROLLUP;
```

This query will group the data by both the region and year columns, and also include summary rows that provide subtotals for each level of grouping. The output of this query might look something like this:

```
+--------+------+-------------+
| region | year | total_sales |
+--------+------+-------------+
| East   | 2018 |     5000    |
| East   | 2019 |     6000    |
| East   | NULL |    11000    |
| West   | 2018 |     4000    |
| West   | 2019 |     5500    |
| West   | NULL |     9500    |
| NULL   | NULL |    20500    |
+--------+------+-------------+
```