GROUP BY and HAVING

This lesson shows how to use the GROUP BY and HAVING commands in SQL to group data. This is a useful way to summarize and understand large datasets.

Although large datasets can contain valuable information, understanding and summarizing the data in them often requires a few extra steps. When you get a new dataset, your first instinct may be to scroll through the data to familiarize yourself with it.

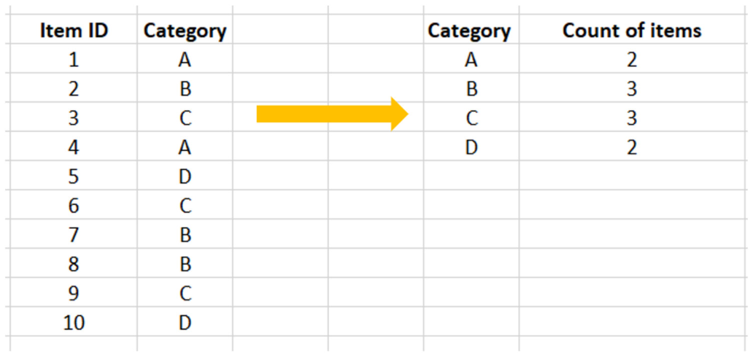
This is a good habit and should always be done when you first encounter data—but it doesn't tell you much about the aggregate data.

Scrolling shows you what columns there are, what data types exist, and what some of the values look like. But it doesn't tell you how many rows share a certain type of attribute or how many groups of attributes there are. Aggregating, or grouping, data is the best way to get this information.

Aggregate data

Grouped or summarized data that is created by combining multiple rows of individual-level data.

You can group data by attributes that are repeated in multiple rows. For example, the data below contains item IDs and the categories that they fall into. Notice that the table on the left has repeated attributes: categories A, B, C, and D. The table on the right summarizes that data by treating each category as a group. In other words, the data in the right table is summarized; it has been grouped by category.



Item ID	Category
1	A
2	B
3	C
4	A
5	D
6	C
7	B
8	B
9	C
10	D

Category	Count of items
A	2
B	3
C	3
D	2

Seeing data in a grouped view is useful because it can help you understand it better. In the example above, you can see exactly how many items are in each category. You can use this to answer other questions: What percentage of items are in each category? How many categories are there?

Think back to when you learned about pivot tables in Excel. Why are pivot tables so useful? It's because they can quickly show you aggregated, or grouped, summary information. The GROUP BY command is SQL's version of a pivot table. GROUP BY summarizes data by treating repeating values as groups and reorganizing the data into them.

GROUP BY to see column values

The simplest use of GROUP BY is to see all possible values in a column. For example, if you have a column that lists various years, it would be helpful to know which year values are included; that way, you'll know which years the data covers. You could use GROUP BY to see all the different year values in the column.

The overall structure for GROUP BY is as follows:

```sql
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

The columns after SELECT are the columns that you want to see in your output, in the order that they're listed. The WHERE clause is optional and is used to add a filter based on a condition. The columns after GROUP BY are the columns that you want to group the data by.

GROUP BY to summarize data

The other use of GROUP BY is to summarize a dataset by grouping the data in a specified way. The same GROUP BY structure applies; it looks like this:

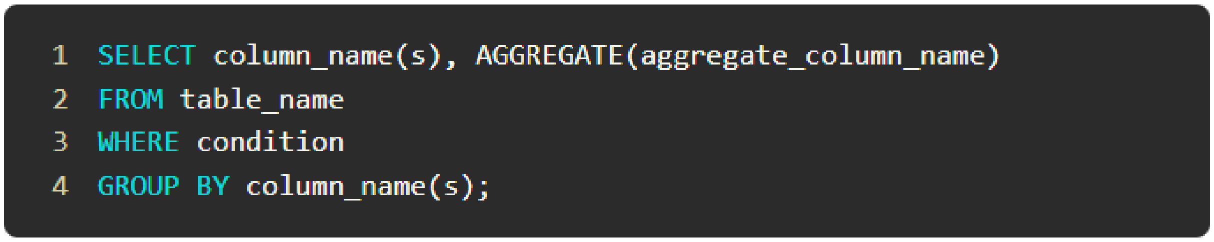
```sql
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

In the examples so far, you selected the column that you grouped by to see all possible values for that column. But what if you want to pull multiple columns to see how they are grouped?

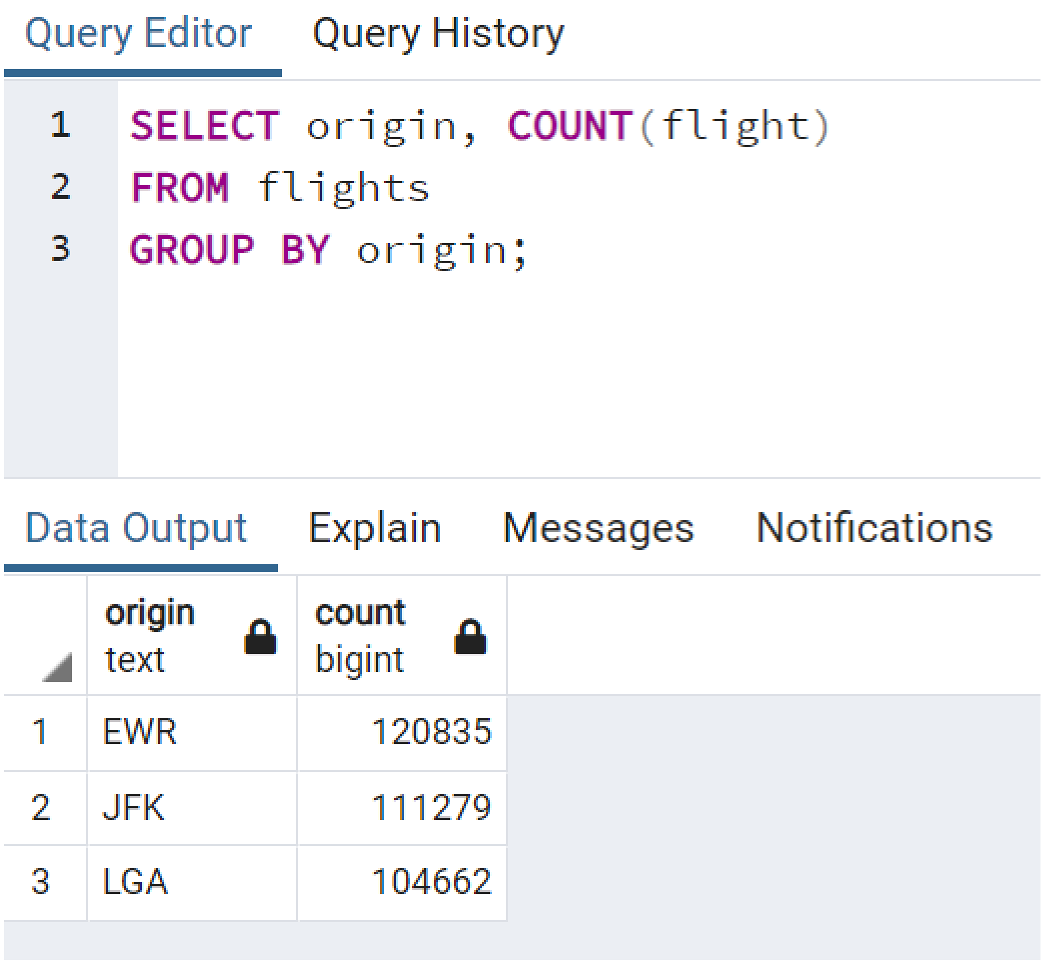
*Unfortunately, when using GROUP BY, you cannot select all columns with *. This is because GROUP BY reorganizes the data by moving it into groups. This process changes the layout of the columns, so the * shortcut results in an error.*

To understand why this happens, revisit the diagram that you saw at the beginning of this lesson. As you can see below, the first table has an Item ID column. But the Item ID column doesn't exist in the second table because the data has been grouped. Instead, Item ID is summarized as Count of items because it shows how many Item IDs there are in each group. This is why you cannot use SELECT * with GROUP BY: many of the columns don't exist in their original forms anymore.

So, to select multiple columns, write out the aggregate forms of the columns that you want to view, separated by commas. The aggregate form of a column is a column with an aggregate function applied to it. Recall that the aggregate functions in SQL include **SUM**, **AVG**, **COUNT**, **MIN**, and **MAX**.



```sql
SELECT column_name(s), AGGREGATE(aggregate_column_name)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```



```sql
SELECT origin, COUNT(flight)
FROM flights
GROUP BY origin;
```

	origin text	count bigint
1	EWR	120835
2	JFK	111279
3	LGA	104662

HAVING

One thing that you may want to add to a GROUP BY clause is a condition. By now, you're used to using WHERE to add a condition to filter the data—but unfortunately, WHERE cannot be applied to aggregate data. So, to add a condition to GROUP BY, use the HAVING clause.

The clause structure for HAVING is as follows:

```sql
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition;
```

HAVING is similar to WHERE; however, with HAVING, you must specify the aggregate condition with it. As always, you can add a WHERE clause here if you also want to add a nonaggregated filter to your data selection.

Recall the last query that you ran, when you found the number of flights leaving each origin airport. Try adding a condition to it so that only origin airports with over 111,000 flights show up. You can use the following query:

```sql
SELECT origin, COUNT(flight)
FROM flights
GROUP BY origin
HAVING COUNT(flight) > 111000;
```

Notice that the condition after HAVING is an aggregate condition—it's applied to the total count of flights—but the way that it's used is similar to WHERE.

The output of this query is shown below. The output now includes only EWR and JFK airports; it excludes the LGA airport because LGA has fewer than 111,000 departures.