Ordering, limiting, and grouping tutorial

In this tutorial, you'll write SQL queries that demonstrate:

- How to specify the sort order of the query results
- How to restrict the number of query results
- How to combine data by summing, counting, and other ways of aggregating it
- How to include aggregated data for subsets within the query results
- How to join strings together in query results

To get started, open the `ordering-limiting-grouping-tutorial.sql` file in pgAdmin. You'll add the queries for this tutorial under the corresponding sections. All these queries use the `PizzaShop` database.

Part one: ORDER BY clauses

Start by writing a query to retrieve the names of all the pizza toppings:

```
SELECT topping_name
FROM topping;
```

Run that query, and notice that the results don't appear to be in any particular order. To sort them alphabetically, add an `ORDER BY` clause:

```
SELECT topping_name
FROM topping
ORDER BY topping_name;
```

> Note: the first query might return results in alphabetical order, but there's no guarantee that it'll always return in the same order. If you want to be sure of the order, use the `ORDER BY` clause.

When you run that query, you'll see that "Anchovies" appears first, since it begins with "A", followed by "Bacon", and so on through "Tomatoes." By default, `ORDER BY` sorts the results in *ascending* order. Switch it to *descending* order by adding `DESC` after the field name:

```
SELECT topping_name
FROM topping
ORDER BY topping_name DESC;
```

Run it, and notice that "Tomatoes" is now the first result and "Anchovies" is last.

Part two: LIMIT clauses

Write a query to retrieve the ids and amounts of all the shop's sales, sorted from highest to lowest:

```
SELECT sale_id, total
FROM sale
ORDER BY total DESC;
```

Running that query returns as many results as there are records in the `sale` table. To see the top five sales instead of all of them, add a `LIMIT` clause:

```
SELECT sale_id, total
FROM sale
ORDER BY total DESC
LIMIT 5;
```

Run that query, and notice there are only five rows in the result set. Those are the top five sales because of the `ORDER BY` clause.

Part three: Aggregate functions

To retrieve the combined total of all the shop's sales, write a query with the `SUM` function:

```
SELECT SUM(total) AS total_sales
FROM sale;
```

Run it, and you'll see a sum of 1248.80. To get the combined total for one customer, add a `WHERE` clause:

```
SELECT SUM(total) AS total_sales
FROM sale
WHERE customer_id = 37;
```

Now the query returns a sum of 48.21, by adding up the four sales made to the customer with an id of 37.

Three other aggregate functions that are useful when working with numbers are `MIN`, `MAX`, and `AVG`. Write a query that returns the shop's smallest sale, largest sale, and average sale:

```
SELECT MIN(total) AS min_sales, MAX(total) AS max_sales, AVG(total) AS avg_sales
FROM sale;
```

Run it, and you'll see that the smallest sale is 9.99, the largest is 44.22, and the average is 17.3444444444444444. To round the average to two decimal places, use the `ROUND` function:

```
SELECT MIN(total) AS min_sales, MAX(total) AS max_sales, ROUND(AVG(total), 2) AS
avg_sales
```

```
FROM sale;
```

Running that query returns an average sale of 17.34.

One more commonly used aggregate function is `COUNT`. Write a query to count how many pizzas had pineapple on them:

```
SELECT COUNT(*) AS times_used
FROM pizza_topping
WHERE topping_name = 'Pineapple';
```

Run the query, and you'll see the number of pizzas with pineapple is five.

Part four: GROUP BY statements

To find out how many pizzas there were with each of the possible toppings, write a query using `GROUP BY`:

```
SELECT topping_name, COUNT(*) AS times_used
FROM pizza_topping
GROUP BY topping_name;
```

After running the query, look at the results. Notice that they're either sorted by topping name, or not sorted in any meaningful way.

Add an `ORDER BY` clause to sort them from the most frequently used topping to the least frequently used, and give the count the alias `times_used`:

```
SELECT topping_name, COUNT(*) AS times_used
FROM pizza_topping
GROUP BY topping_name
ORDER BY times_used DESC;
```

Now when you run it, you'll see the results are sorted with Bacon and Tomatoes at the top, each of which was used 11 times.

Part five: String concatenation

The `customer` table contains each customer's first name and last name. Write a query that retrieves the names of all the customers in alphabetical order by their last name:

```
SELECT first_name, last_name
FROM customer
ORDER BY last_name;
```

Run that query, and you'll see that each customer's first name and last name are in separate columns. To join them together into a single column, use the `||` operator between the field names:

```
SELECT first_name || last_name AS full_name
FROM customer
ORDER BY last_name;
```

Now that query returns one column, but the first and last names are run together without any space between them, and the column has no name. Add another `||` operator and a literal space to make it look better, and label the single column `full_name`:

```
SELECT first_name || ' ' || last_name AS full_name
FROM customer
ORDER BY last_name;
```

When you run this query, it looks better. Notice that the results are still sorted by the last name field, even though it no longer appears in its own column.

Next steps

If you'd like to practice writing more queries that involve ordering, limiting, and grouping, here are a few ideas:

- Find out how many customers provided their email address. (Hint: if you specify a field name in the parentheses after `COUNT` rather than `COUNT(*)`, only the fields with non-null values are counted)
- The examples in this tutorial only used a single field with `ORDER BY`, but you can specify as many fields as you want. Try adding more, and see what happens. (Hint: look at how records are sorted where the first field has the same value)
- You can also use `GROUP BY` with multiple fields. Try writing a query for the `pizza` table that groups by crust and sauce.