# Ordering, grouping, and aggregate functions

The purpose of this exercise is to practice using ordering, grouping, and aggregate functions to summarize data using Structured Query Language (SQL).

## Learning objectives

After completing this exercise, you'll understand:

- How to order SQL query results using the `ORDER BY` statement.
- How to limit SQL query results using the `LIMIT` statement.
- How to concatenate strings together in SQL.
- How to use aggregate functions to summarize multiple rows of data.
- How to use the `GROUP BY` statement.

## Evaluation criteria and functional requirements

- All of the queries run as expected.
- The number of results returned from your query equals the number of results specified in each question.
- The unit tests pass as expected.
- Code is clean, concise, and readable.
- You shouldn't need to use sub-queries to produce the expected results.

To complete this exercise, you need to write SQL queries in the files that are in the `Exercises` folder. You'll use the `UnitedStates` database as a source for all queries.

In each file, there's a commented out problem statement. Below it, write the query needed to solve the problem. The value immediately after the problem statement is the expected number of rows that the query must return. Some of the queries must only return one row and column.

In some files, there's a hint that tells you that the expected value is around a certain number. For example, if your query must return `68117`, the hint reads "Expected answer is around 68,000."

## Getting started

1. If you haven't done so already, create the `UnitedStates` database. The script for this is available in yesterday's lecture code.
2. Open the `Exercises` folder. Each file is numbered in suggested order of completion, but you can do them in any order you wish.
3. To start, double click any file to open it in DbVisualizer and write the query. Then, double click another exercise file to continue.
    - Alternatively, once in DbVisualizer you can open files using the menu option **File -> Open...**.
4. The unit tests project is in the same directory as this README. You can open it in IntelliJ and run the tests as you did in earlier exercises.

> Note: Make sure to save your changes to the SQL file before running the unit tests.

## Tips and tricks

- The `ORDER BY` statement orders results based on the value in a column. By default, the results are sorted in ascending order. However, if you want to sort in descending order, you can add the `DESC` keyword after the column name. You can also explicitly state that the sort order is ascending using the `ASC` keyword.
- The `LIMIT` statement can be helpful when you want to reduce the size of the results to a specific number. For instance, if you only want to get five of the rows from a result set, you can do so by specifying `LIMIT 5` at the end of your query. Essentially, the results are truncated when using this statement. Keep in mind that you should typically combine this with an `ORDER BY` statement if you're looking for the top or bottom results. The PostgreSQL documentation explains why ordering is important in more detail.
- Occasionally, you'll need to combine multiple fields to represent data. This can be achieved using a concatenation operator (`||`). For instance, given a table of employees, if you wanted to get the full name for all of the employees in the table, you could write the following query:

```
SELECT first_name || ' ' || last_name as employee_name
FROM employee
```

- Often, you may need to aggregate data when writing SQL queries. PostgreSQL offers several aggregate functions that can be useful for summarizing and grouping data. Several functions used often include:
  - `AVG` returns the average value of a numeric column
  - `SUM` returns the total sum of a numeric column
  - `COUNT` returns the number of rows matching criteria
  - `MIN` returns the smallest value of the selected column
  - `MAX` returns the largest value of the selected column
- Sometimes when using aggregate functions, you need to `GROUP BY` other columns to get multiple rows of results. Without the `GROUP BY` clause, you'll only receive a single result row from your query. To get multiple rows, you must specify the fields that duplicate values should be removed for. For instance, if you wanted to get the total sales that each employee had made ordered by the employee ID, you might write a query like the following:

```
SELECT employee_id, SUM(sales_amount)
FROM employee_sale
GROUP BY employee_id
ORDER BY employee_id;
```

---