**Filtering results**

Congrats on finishing the first chapter! You now know how to select columns and perform basic counts. This chapter will focus on filtering your results.

In SQL, the WHERE keyword allows you to filter based on both text and numeric values in a table. There are a few different comparison operators you can use:

* = equal
* <> not equal
* < less than
* > greater than
* <= less than or equal to
* >= greater than or equal to

For example, you can filter text records such as title. The following code returns all films with the title 'Metropolis':

SELECT title

FROM films

WHERE title = 'Metropolis';

Notice that the WHERE clause always comes after the FROM statement!

**Note that in this course we will use <> and not != for the not equal operator, as per the SQL standard.**

What does the following query return?

SELECT title

FROM films

WHERE release\_year > 2000;

# WHERE AND OR

What if you want to select rows based on multiple conditions where some but not all of the conditions need to be met? For this, SQL has the OR operator.

For example, the following returns all films released in either 1994 or 2000:

SELECT title

FROM films

WHERE release\_year = 1994

OR release\_year = 2000;

Note that you need to specify the column for every OR condition, so the following is invalid:

SELECT title

FROM films

WHERE release\_year = 1994 OR 2000;

When combining AND and OR, be sure to enclose the individual clauses in parentheses, like so:

SELECT title

FROM films

WHERE (release\_year = 1994 OR release\_year = 1995)

AND (certification = 'PG' OR certification = 'R');

Otherwise, due to SQL's precedence rules, you may not get the results you're expecting!

What does the OR operator do?

# BETWEEN

As you've learned, you can use the following query to get titles of all films released in and between 1994 and 2000:

SELECT title

FROM films

WHERE release\_year >= 1994

AND release\_year <= 2000;

Checking for ranges like this is very common, so in SQL the BETWEEN keyword provides a useful shorthand for filtering values within a specified range. This query is equivalent to the one above:

SELECT title

FROM films

WHERE release\_year

BETWEEN 1994 AND 2000;

It's important to remember that BETWEEN is inclusive, meaning the beginning and end values are included in the results!

What does the BETWEEN keyword do?

# WHERE IN

As you've seen, WHERE is very useful for filtering results. However, if you want to filter based on many conditions, WHERE can get unwieldy. For example:

SELECT name

FROM kids

WHERE age = 2

OR age = 4

OR age = 6

OR age = 8

OR age = 10;

Enter the IN operator! The IN operator allows you to specify multiple values in a WHERE clause, making it easier and quicker to specify multiple OR conditions! Neat, right?

So, the above example would become simply:

SELECT name

FROM kids

WHERE age IN (2, 4, 6, 8, 10);

Try using the IN operator yourself!

# Introduction to NULL and IS NULL

In SQL, NULL represents a missing or unknown value. You can check for NULL values using the expression IS NULL. For example, to count the number of missing birth dates in the people table:

SELECT COUNT(\*)

FROM people

WHERE birthdate IS NULL;

As you can see, IS NULL is useful when combined with WHERE to figure out what data you're missing.

Sometimes, you'll want to filter out missing values so you only get results which are not NULL. To do this, you can use the IS NOT NULL operator.

For example, this query gives the names of all people whose birth dates are not missing in the people table.

SELECT name

FROM people

WHERE birthdate IS NOT NULL;

# LIKE and NOT LIKE

As you've seen, the WHERE clause can be used to filter text data. However, so far you've only been able to filter by specifying the exact text you're interested in. In the real world, often you'll want to search for a pattern rather than a specific text string.

In SQL, the LIKE operator can be used in a WHERE clause to search for a pattern in a column. To accomplish this, you use something called a wildcard as a placeholder for some other values. There are two wildcards you can use with LIKE:

The % wildcard will match zero, one, or many characters in text. For example, the following query matches companies like 'Data', 'DataC' 'DataCamp', 'DataMind', and so on:

SELECT name

FROM companies

WHERE name LIKE 'Data%';

The \_ wildcard will match a single character. For example, the following query matches companies like 'DataCamp', 'DataComp', and so on:

SELECT name

FROM companies

WHERE name LIKE 'DataC\_mp';

You can also use the NOT LIKE operator to find records that don't match the pattern you specify.

# Aggregate functions

Often, you will want to perform some calculation on the data in a database. SQL provides a few functions, called aggregate functions, to help you out with this.

For example,

SELECT AVG(budget)

FROM films;

gives you the average value from the budget column of the films table. Similarly, the MAX function returns the highest budget:

SELECT MAX(budget)

FROM films;

The SUM function returns the result of adding up the numeric values in a column:

SELECT SUM(budget)

FROM films;

You can probably guess what the MIN function does! Now it's your turn to try out some SQL functions.

# Combining aggregate functions with WHERE

Aggregate functions can be combined with the WHERE clause to gain further insights from your data.

For example, to get the total budget of movies made in the year 2010 or later:

SELECT SUM(budget)

FROM films

WHERE release\_year >= 2010;

Now it's your turn!

# A note on arithmetic

In addition to using aggregate functions, you can perform basic arithmetic with symbols like +, -, \*, and /.

So, for example, this gives a result of 12:

SELECT (4 \* 3);

However, the following gives a result of 1:

SELECT (4 / 3);

What's going on here?

SQL assumes that if you divide an integer by an integer, you want to get an integer back. So be careful when dividing!

If you want more precision when dividing, you can add decimal places to your numbers. For example,

SELECT (4.0 / 3.0) AS result;

gives you the result you would expect: 1.333.

# It's AS simple AS aliasing

You may have noticed in the first exercise of this chapter that the column name of your result was just the name of the function you used. For example,

SELECT MAX(budget)

FROM films;

gives you a result with one column, named max. But what if you use two functions like this?

SELECT MAX(budget), MAX(duration)

FROM films;

Well, then you'd have two columns named max, which isn't very useful!

To avoid situations like this, SQL allows you to do something called aliasing. Aliasing simply means you assign a temporary name to something. To alias, you use the AS keyword, which you've already seen earlier in this course.

For example, in the above example we could use aliases to make the result clearer:

SELECT MAX(budget) AS max\_budget,

MAX(duration) AS max\_duration

FROM films;

Aliases are helpful for making results more readable!

# Even more aliasing

Let's practice your newfound aliasing skills some more before moving on!

**Recall:** SQL assumes that if you divide an integer by an integer, you want to get an integer back.

This means that the following will erroneously result in 400.0:

SELECT 45 / 10 \* 100.0;

This is because 45 / 10 evaluates to an integer (4), and not a decimal number like we would expect.

So when you're dividing make sure at least one of your numbers has a decimal place:

SELECT 45 \* 100.0 / 10;

The above now gives the correct answer of 450.0 since the numerator (45 \* 100.0) of the division is now a decimal!

# ORDER BY

Congratulations on making it this far! You now know how to select and filter your results.

In this chapter you'll learn how to sort and group your results to gain further insight. Let's go!

In SQL, the ORDER BY keyword is used to sort results in ascending or descending order according to the values of one or more columns.

By default ORDER BY will sort in ascending order. If you want to sort the results in descending order, you can use the DESC keyword. For example,

SELECT title

FROM films

ORDER BY release\_year DESC;

gives you the titles of films sorted by release year, from newest to oldest.

How do you think ORDER BY sorts a column of text values by default?

# Sorting single columns (DESC)

To order results in descending order, you can put the keyword DESC after your ORDER BY. For example, to get all the names in the people table, in reverse alphabetical order:

SELECT name

FROM people

ORDER BY name DESC;

Now practice using ORDER BY with DESC to sort single columns in descending order!

# Sorting multiple columns

ORDER BY can also be used to sort on multiple columns. It will sort by the first column specified, then sort by the next, then the next, and so on. For example,

SELECT birthdate, name

FROM people

ORDER BY birthdate, name;

sorts on birth dates first (oldest to newest) and then sorts on the names in alphabetical order. **The order of columns is important!**

Try using ORDER BY to sort multiple columns! Remember, to specify multiple columns you separate the column names with a comma.

# GROUP BY

Now you know how to sort results! Often you'll need to aggregate results. For example, you might want to count the number of male and female employees in your company. Here, what you want is to group all the males together and count them, and group all the females together and count them. In SQL, GROUP BY allows you to group a result by one or more columns, like so:

SELECT sex, count(\*)

FROM employees

GROUP BY sex;

This might give, for example:

| **sex** | **count** |
| --- | --- |
| male | 15 |
| female | 19 |

Commonly, GROUP BY is used with aggregate functions like COUNT() or MAX(). Note that GROUP BY always goes after the FROM clause!

# GROUP BY practice

As you've just seen, combining aggregate functions with GROUP BY can yield some powerful results!

A word of warning: SQL will return an error if you try to SELECT a field that is not in your GROUP BY clause without using it to calculate some kind of value about the entire group.

Note that you can combine GROUP BY with ORDER BY to group your results, calculate something about them, and then order your results. For example,

SELECT sex, count(\*)

FROM employees

GROUP BY sex

ORDER BY count DESC;

might return something like

| **sex** | **count** |
| --- | --- |
| female | 19 |
| male | 15 |

because there are more females at our company than males. Note also that ORDER BY always goes after GROUP BY. Let's try some exercises!

# GROUP BY practice (2)

Now practice your new skills by combining GROUP BY and ORDER BY with some more aggregate functions!

Make sure to always put the ORDER BY clause at the end of your query. You can't sort values that you haven't calculated yet!

ms

GROUP BY release\_year

HAVING COUNT(title) > 10;

shows only those years in which more than 10 films were released.

# All together now (2)

Great work! Now try another large query. This time, all in one go!

Remember, if you only want to return a certain number of results, you can use the LIMIT keyword to limit the number of rows returned

# Inner join

PostgreSQL was mentioned in the slides but you'll find that these joins and the material here applies to different forms of SQL as well.

Throughout this course, you'll be working with the countries database containing information about the most populous world cities as well as country-level economic data, population data, and geographic data. This countries database also contains information on languages spoken in each country.

You can see the different tables in this database by clicking on the tabs on the bottom right below **query.sql**. Click through them to get a sense for the types of data that each table contains before you continue with the course! Take note of the fields that appear to be shared across the tables.

Recall from the video the basic syntax for an INNER JOIN, here including all columns in **both** tables:

SELECT \*

FROM left\_table

INNER JOIN right\_table

ON left\_table.id = right\_table.id;

You'll start off with a SELECT statement and then build up to an inner join with the cities and countries tables. Let's get to it!

**EXAMPLE:**

SELECT \*

FROM cities

-- 1. Inner join to countries

INNER JOIN countries

-- 2. Match on the country codes

ON cities.country\_code = countries.code;

**EXAMPLE:**

-- 1. Select name fields (with alias) and region

SELECT cities.name AS city, countries.name AS country, region

FROM cities

INNER JOIN countries

ON cities.country\_code = countries.code;

# Inner join (2)

Instead of writing the full table name, you can use table aliasing as a shortcut. For tables you also use AS to add the alias immediately after the table name with a space. Check out the aliasing of cities and countries below.

SELECT c1.name AS city, c2.name AS country

FROM cities AS c1

INNER JOIN countries AS c2

ON c1.country\_code = c2.code;

Notice that to select a field in your query that appears in multiple tables, you'll need to identify which table/table alias you're referring to by using a . in your SELECT statement.

You'll now explore a way to get data from both the countries and economies tables to examine the inflation rate for both 2010 and 2015.

Sometimes it's easier to write SQL code out of order: you write the SELECT statement after you've done the JOIN.

* Join the tables countries (left) and economies (right) aliasing countries AS c and economies AS e.
* Specify the field to match the tables ON.
* From this join, SELECT:
  + c.code, aliased as country\_code.
  + name, year, and inflation\_rate, not aliased.

[**Take Hint (-30 XP)**](javascript:void(0))

**EXAMPLE:**

-- 3. Select fields with aliases

SELECT c.code AS country\_code, c.name, e.year, e.inflation\_rate

FROM countries AS c

-- 1. Join to economies (alias e)

INNER JOIN economies AS e

-- 2. Match on code

ON c.code = e.code;

# Inner join (3)

The ability to combine multiple joins in a single query is a powerful feature of SQL, e.g:

SELECT \*

FROM left\_table

INNER JOIN right\_table

ON left\_table.id = right\_table.id

INNER JOIN another\_table

ON left\_table.id = another\_table.id;

As you can see here it becomes tedious to continually write long table names in joins. This is when it becomes useful to alias each table using the first letter of its name (e.g. countries AS c)! It is standard practice to alias in this way and, if you choose to alias tables or are asked to specifically for an exercise in this course, you should follow this protocol.

Now, for each country, you want to get the country name, its region, and the fertility rate and unemployment rate for both 2010 and 2015.

Note that results should work throughout this course with or without table aliasing unless specified differently.

**EXAMPLE:**

* Inner join countries (left) and populations (right) on the code and country\_code fields respectively.
* Alias countries AS c and populations AS p.
* Select code, name, and region from countries and also select year and fertility\_rate from populations (5 fields in total).

-- 4. Select fields

SELECT c.code, c.name, c.region, p.fertility\_rate, p.year

-- 1. From countries (alias as c)

FROM countries AS c

-- 2. Join with populations (as p)

INNER JOIN populations AS p

-- 3. Match on country code

ON c.code = p.country\_code;

# Inner join (3)

The ability to combine multiple joins in a single query is a powerful feature of SQL, e.g:

SELECT \*

FROM left\_table

INNER JOIN right\_table

ON left\_table.id = right\_table.id

INNER JOIN another\_table

ON left\_table.id = another\_table.id;

As you can see here it becomes tedious to continually write long table names in joins. This is when it becomes useful to alias each table using the first letter of its name (e.g. countries AS c)! It is standard practice to alias in this way and, if you choose to alias tables or are asked to specifically for an exercise in this course, you should follow this protocol.

Now, for each country, you want to get the country name, its region, and the fertility rate and unemployment rate for both 2010 and 2015.

Note that results should work throughout this course with or without table aliasing unless specified differently.

* Add an additional inner join with economies to your previous query by joining on code.
* Include the unemployment\_rate column that became available through joining with economies.
* Note that year appears in both populations and economies, so you have to explicitly use e.year instead of year as you did before.

**EXAMPLE:**

-- 6. Select fields

SELECT c.code, name, region, e.year, fertility\_rate, unemployment\_rate

-- 1. From countries (alias as c)

FROM countries AS c

-- 2. Join to populations (as p)

INNER JOIN populations AS p

-- 3. Match on country code

ON c.code = p.country\_code

-- 4. Join to economies (as e)

INNER JOIN economies AS e

-- 5. Match on country code

ON c.code = e.code;

# Inner join (3)

The ability to combine multiple joins in a single query is a powerful feature of SQL, e.g:

SELECT \*

FROM left\_table

INNER JOIN right\_table

ON left\_table.id = right\_table.id

INNER JOIN another\_table

ON left\_table.id = another\_table.id;

As you can see here it becomes tedious to continually write long table names in joins. This is when it becomes useful to alias each table using the first letter of its name (e.g. countries AS c)! It is standard practice to alias in this way and, if you choose to alias tables or are asked to specifically for an exercise in this course, you should follow this protocol.

Now, for each country, you want to get the country name, its region, and the fertility rate and unemployment rate for both 2010 and 2015.

Note that results should work throughout this course with or without table aliasing unless specified differently.

* Scroll down the query result and take a look at the results for Albania from your previous query. Does something seem off to you?
* The trouble with doing your last join on c.code = e.code and not also including year is that e.g. the 2010 value for fertility\_rate is also paired with the 2015 value for unemployment\_rate.
* Fix your previous query: in your last ON clause, use AND to add an additional joining condition. In addition to joining on code in c and e, also join on year in e and p.

**EXAMPLE:**-- 6. Select fields

SELECT c.code, name, region, e.year, fertility\_rate, unemployment\_rate

-- 1. From countries (alias as c)

FROM countries AS c

-- 2. Join to populations (as p)

INNER JOIN populations AS p

-- 3. Match on country code

ON c.code = p.country\_code

-- 4. Join to economies (as e)

INNER JOIN economies AS e

-- 5. Match on country code and year

ON c.code = e.code AND e.year = p.year

# Review inner join using on

Why does the following code result in an error?

SELECT c.name AS country, l.name AS language

FROM countries AS c

INNER JOIN languages AS l;

# Inner join with using

When joining tables with a common field name, e.g.

SELECT \*

FROM countries

INNER JOIN economies

ON countries.code = economies.code

You can use USING as a shortcut:

SELECT \*

FROM countries

INNER JOIN economies

USING(code)

You'll now explore how this can be done with the countries and languages tables.

* Inner join countries on the left and languages on the right with USING(code).
* Select the fields corresponding to:
  + country name AS country,
  + continent name,
  + language name AS language, and
  + whether or not the language is official.

Remember to alias your tables using the first letter of their names.

**EXAMPLE:**

-- 4. Select fields

SELECT c.name AS country, continent, l.name AS language, official

-- 1. From countries (alias as c)

FROM countries AS c

-- 2. Join to languages (as l)

INNER JOIN languages AS l

-- 3. Match using code

USING(code);

# Self-join

In this exercise, you'll use the populations table to perform a self-join to calculate the percentage increase in population from 2010 to 2015 for each country code!

Since you'll be joining the populations table to itself, you can alias populations as p1 and also populations as p2. This is good practice whenever you are aliasing and your tables have the same first letter. Note that you are required to alias the tables with self-joins.

