**Joining Data in SQL**

# 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!

------------------------------------------------------

Begin by selecting all columns from the cities table.

-- Select all columns from cities

select \*

from cities

------------------------------------------------------

* Inner join the cities table on the left to the countries table on the right, keeping all of the fields in both tables.
* You should match the tables on the country\_code field in cities and the code field in countries.
* **Do not** alias your tables here or in the next step. Using cities and countries is fine for now.

SELECT \*

FROM cities

-- 1. Inner join to countries

INNER JOIN countries

-- 2. Match on the country codes

ON cities.country\_code = countries.code;

* Modify the SELECT statement to keep only the name of the city, the name of the country, and the name of the region the country resides in.
* Recall from our [**Intro to SQL for Data Science**](https://www.datacamp.com/courses/intro-to-sql-for-data-science) course that you can alias fields using AS. Alias the name of the city AS city and the name of the country AS country.

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.

-- 3. Select fields with aliases

SELECT c.code AS country\_code, name, year, 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.

* 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 code, name, region, year, fertility\_rate

-- 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

* 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.

-- 6. Select fields

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

* 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 cand e, also join on year in e and p.

-- 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;

# 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.

-- 4. Select fields

select c.name AS country, c.continent, l.name AS language, l.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.

* Join populations with itself ON country\_code.
* Select the country\_code from p1 and the size field from both p1 and p2. SQL won't allow same-named fields, so alias p1.size as size2010 and p2.size as size2015.

-- 4. Select fields with aliases

select p1.country\_code,

p1.size as size2010,

p2.size as size2015

-- 1. From populations (alias as p1)

from populations as p1

-- 2. Join to itself (alias as p2)

INNER JOIN populations as p2

-- 3. Match on country code

ON p1.country\_code=p2.country\_code

Notice from the result that for each country\_code you have four entries laying out all combinations of 2010 and 2015.

* Extend the ON in your query to include only those records where the p1.year (2010) matches with p2.year - 5 (2015 - 5 = 2010). This will omit the three entries per country\_code that you aren't interested in.

-- 5. Select fields with aliases

SELECT p1.country\_code,

p1.size AS size2010,

p2.size AS size2015

-- 1. From populations (alias as p1)

FROM populations as p1

-- 2. Join to itself (alias as p2)

INNER JOIN populations as p2

-- 3. Match on country code

ON p1.country\_code = p2.country\_code

-- 4. and year (with calculation)

AND p1.year = p2.year - 5;

As you just saw, you can also use SQL to calculate values like p2.year - 5 for you. With two fields like size2010 and size2015, you may want to determine the percentage increase from one field to the next:

With two numeric fields A and B, the percentage growth from A to B can be calculated as (B−A)/A∗100.0.

Add a new field to SELECT, aliased as growth\_perc, that calculates the percentage population growth from 2010 to 2015 for each country, using p2.size and p1.size.

SELECT p1.country\_code,

p1.size AS size2010,

p2.size AS size2015,

-- 1. calculate growth\_perc

((p2.size - p1.size)/p1.size \* 100.0) AS growth\_perc

-- 2. From populations (alias as p1)

FROM populations AS p1

-- 3. Join to itself (alias as p2)

INNER JOIN populations AS p2

-- 4. Match on country code

ON p1.country\_code = p2.country\_code

-- 5. and year (with calculation)

AND p1.year = p2.year - 5;

Using the countries table, create a new field AS geosize\_group that groups the countries into three groups:

* If surface\_area is greater than 2 million, geosize\_group is 'large'.
* If surface\_area is greater than 350 thousand but not larger than 2 million, geosize\_group is 'medium'.
* Otherwise, geosize\_group is 'small'.

SELECT name, continent, code, surface\_area,

-- 1. First case

CASE WHEN surface\_area > 2000000 THEN 'large'

-- 2. Second case

WHEN surface\_area > 350000 THEN 'medium'

-- 3. Else clause + end

ELSE 'small' END

-- 4. Alias name

AS geosize\_group

-- 5. From table

FROM countries;

# Inner challenge

The table you created with the added geosize\_group field has been loaded for you here with the name countries\_plus. Observe the use of (and the placement of) the INTO command to create this countries\_plus table:

SELECT name, continent, code, surface\_area,

CASE WHEN surface\_area > 2000000

THEN 'large'

WHEN surface\_area > 350000

THEN 'medium'

ELSE 'small' END

AS geosize\_group

INTO countries\_plus

FROM countries;

You will now explore the relationship between the size of a country in terms of surface area and in terms of population using grouping fields created with CASE.

By the end of this exercise, you'll be writing two queries back-to-back in a single script. You got this!

Using the populations table focused only for the year 2015, create a new field aliased as popsize\_group to organize population size into

* 'large' (> 50 million),
* 'medium' (> 1 million), and
* 'small' groups.

Select only the country code, population size, and this new popsize\_group as fields.

SELECT country\_code, size,

-- 1. First case

CASE WHEN size > 50000000 THEN 'large'

-- 2. Second case

WHEN size > 1000000 THEN 'medium'

-- 3. Else clause + end

ELSE 'small' END

-- 4. Alias name (popsize\_group)

AS popsize\_group

-- 5. From table

FROM populations

-- 6. Focus on 2015

WHERE year = 2015;

* Use INTO to save the result of the previous query as pop\_plus. You can see an example of this in the countries\_plus code in the assignment text. Make sure to include a ; at the end of your WHERE clause!
* Then, include another query below your first query to display all the records in pop\_plus using SELECT \* FROM pop\_plus; so that you generate results and this will display pop\_plus in **query result**.

SELECT country\_code, size,

CASE WHEN size > 50000000 THEN 'large'

WHEN size > 1000000 THEN 'medium'

ELSE 'small' END

AS popsize\_group

-- 1. Into table

INTO pop\_plus

FROM populations

WHERE year = 2015;

-- 2. Select all columns of pop\_plus

select \* FROM pop\_plus;

* Keep the first query intact that creates pop\_plus using INTO.
* Write a query to join countries\_plus AS c on the left with pop\_plus AS p on the right matching on the country code fields.
* Sort the data based on geosize\_group, in ascending order so that large appears on top.
* Select the name, continent, geosize\_group, and popsize\_group fields.

SELECT country\_code, size,

CASE WHEN size > 50000000

THEN 'large'

WHEN size > 1000000

THEN 'medium'

ELSE 'small' END

AS popsize\_group

INTO pop\_plus

FROM populations

WHERE year = 2015;

-- 5. Select fields

select c.name, c.continent, c.geosize\_group, p.popsize\_group

-- 1. From countries\_plus (alias as c)

from countries\_plus AS c