**2. Joining Data in SQL**

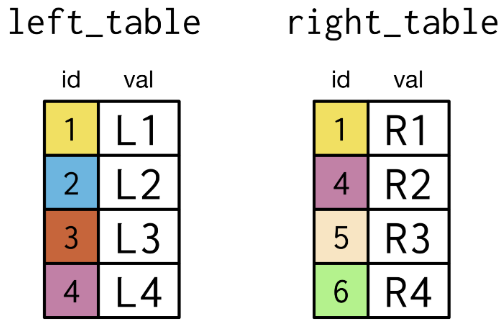
Juntar duas ou mais tabelas em uma única tabela.

INNER JOIN e LEFT JOIN são as junções mais comuns.

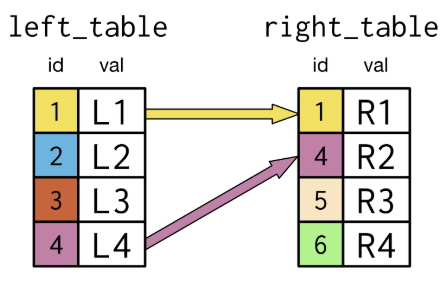
Vamos começar com o layout de um dado e como o INNER JOIN pode ser aplicado a esse dado.

Nos próximos exercícios trabalharemos com duas tabelas nomeadas de left e right. Veja que os valores de mesmo id estão coloridos com a mesma cor.

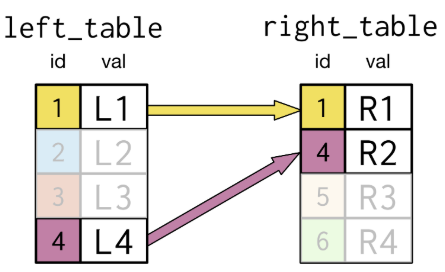
O campo “id” é conhecido como um campo-chave já q ele pode ser usado p/ se referir a um tabela para outra tabela. Ambas as tabelas left e right possuem um campo chamado “val”, que será útil p/ ver quais registros e valores são incluídos em cada join.



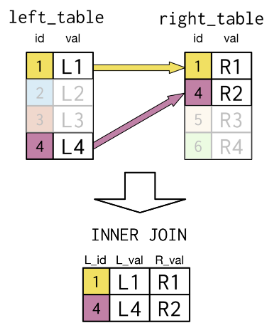
O INNER JOIN apenas inclui registros nos quais a chave está em ambas as tabelas. Veja que aqui o campo id combina apenas os valores 1 e 4. Com INNER JOIN nós procuramos por combinações na tabela right correspondendo a todas as entradas no campo-chave da tabela left.



O foco aqui muda para apenas os registros com uma correspondência em termos do campo “id”. Os regsitros que não são do interesse do INNER JOIN foram desbotados.



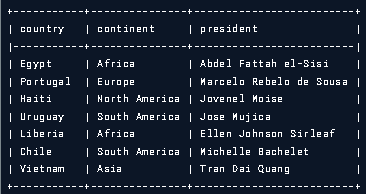
Está é a tabela única resultante da cláusula INNER JOIN que fornece o campo “val” da tabela right com registros correspondentes a apenas aqueles com o valor de id 1 e 4.



Agora que temos noção de como o INNER JOIN funciona, vamos tentar um exemplo em SQL. A tabela prime\_ministers é uma das tabelas da base de dados dos líderes. Note que os países estão inclusos. Suponha que vc esteja interessado em determinar nações que possuem ambos um primeiro ministro e presidente e colocar os resultados em uma tabela só.

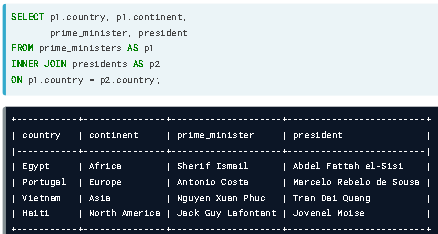


Agora vamos ver a tabela dos presidents. Há países que aparecem em ambas as tabelas. Com tabelas curtas como estas é fácil de ver que Egito, Portugal, Vietnan e Haiti aparecem em ambas as tabelas. Para tabelas grandes, não dá para fazer isso visualmente.

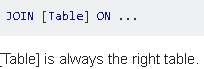


Então usamos a sintaxe do SQL para pegar os países que possuem prime ministers e presidentes dessas duas tabelas para uma.

A sintaxe para completar um INNER JOIN d tabela prime\_ministers para a tabela presidents baseado no campo-chave country é mostrada abaixo. A declaração SELECT é usada para selecionar campos específicos das duas tabelas. Nesse caso, já que countries existe nas duas tabelas, devemos escrever o nome da tabela “ponto” primeiro e depois o nome da coluna (p1.country) para evitar erros. Em seguida listamos a tabela a esquerda do INNER JOIN depois do FROM. E depois listamos a tabela direita depois do INNER JOIN. Por fim, especificamos as chaves das duas tabelas que gostaríamos de combinar.

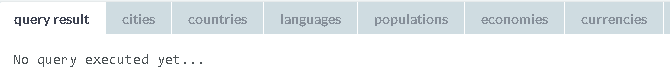


P.S.: Left Table é a primeira tabela no SELECT. Right Table sempre vai ser a tabela que vc está juntando.

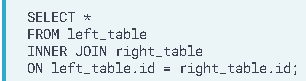


**INNER JOIN**

Vamos usar o PostgreSQL, mas esses JOINs podem ser aplicados para adiferentes formas de SQL. Ao longo do curso vamos usar as bases de dados countries, que contém informações das cidades mais populosas do mundo. Estas são as tabelas que usaremos.

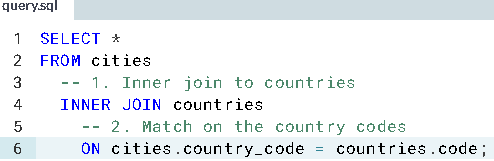


Sintaxe do INNER JOIN:

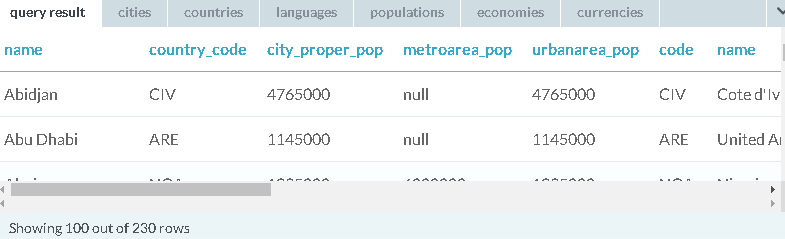


Você começará com uma instrução SELECT e criará uma inner join (junção interna) com as tabelas de cidades e países:

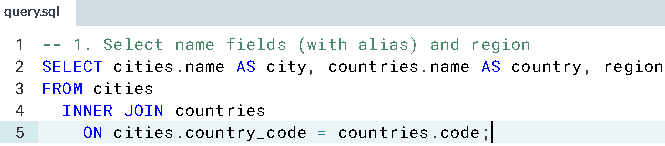
Passo 1: INNER JOIN (associe) a tabela de cities à esquerda a tabela de countries à direita, mantendo todos os campos nas duas tabelas. Você deve corresponder as tabelas no campo country\_code nas cidades e o campo code nos países. Não aliase suas tabelas aqui ou na próxima etapa. Usar cidades e países está bom por enquanto.

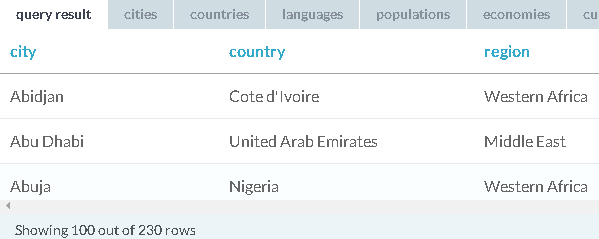


O resultado é a junção da tabela countries a tabela cities.



Passo 2: Modifique a instrução SELECT para manter apenas o nome da cidade, o nome do país e o nome da região em que o país reside. Lembre-se do nosso curso Introdução ao SQL for Data Science que você pode criar alias de campos usando o AS. Aliase o nome da cidade como cidade e o nome do país como país.

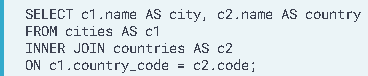




Observe que ao selecionar a coluna region não precisei falar de qual tabela, pois só há essa coluna em uma das tabelas, então o SQL não se confundirá.

**INNER JOIN (2)**

Em vez de escrever o nome completo da tabela, você pode usar o alias da tabela como atalho. Para tabelas, você também usa o AS para adicionar o alias imediatamente após o nome da tabela com um espaço. Confira o alias de cidades e países abaixo.

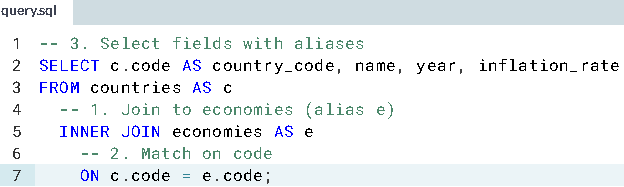


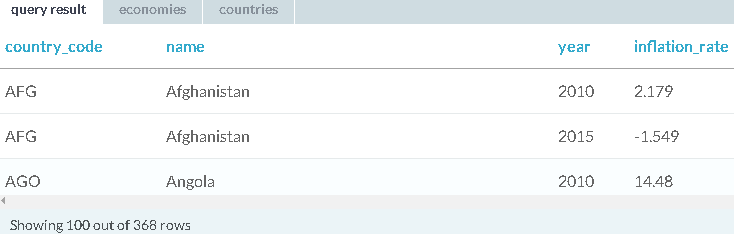
Observe que para selecionar um campo em sua consulta que apareça em várias tabelas, você precisará identificar a que tabela / alias de tabela se refere usando um “.” na sua instrução SELECT.

Agora, você explorará uma maneira de obter dados das tabelas countries e economies para examinar a taxa de inflação em 2010 e 2015.

P.S.: Às vezes, é mais fácil escrever código SQL fora de ordem: você escreve a instrução SELECT depois de fazer JOIN.

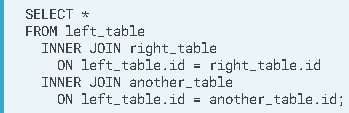
Ex. Junte-se às tabelas countries (esquerda) e economies (direita) aliasing countries AS c e economies AS e. Especifique o campo para corresponder às tabelas ON. Nesta junção, SELECT: c.code, aliased como country\_code. E name, year e inflation\_rate, sem alias:





**INNER JOIN (3)**

A capacidade de combinar várias junções (JOIN) em uma única consulta é um recurso poderoso do SQL, por exemplo:



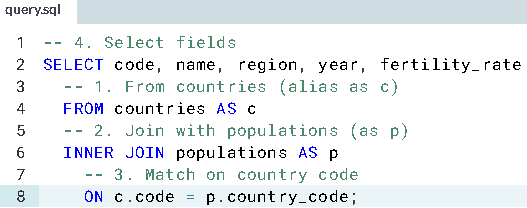
Como você pode ver aqui, torna-se entediante escrever continuamente nomes longos de tabela em junções. Por isso é útil criar um alias para cada tabela usando a primeira letra do nome (por exemplo, countries AS c)! É uma prática padrão usar o alias dessa maneira e, se você optar pelo alias de tabelas ou for solicitado especificamente a um exercício neste curso, você deve seguir este protocolo.

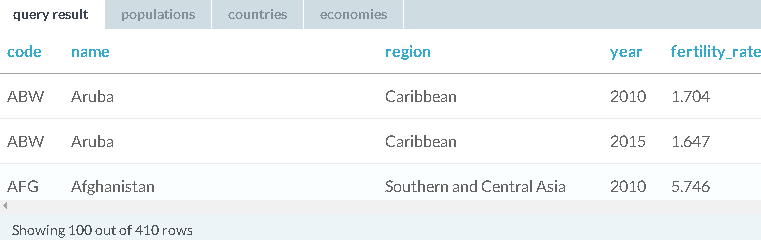
Agora, para cada país, você deseja obter o nome do país, sua região e a taxa de fertilidade e taxa de desemprego de 2010 e 2015.

Observe que os resultados devem funcionar ao longo deste curso com ou sem alias de tabela, a menos que especificado de maneira diferente.

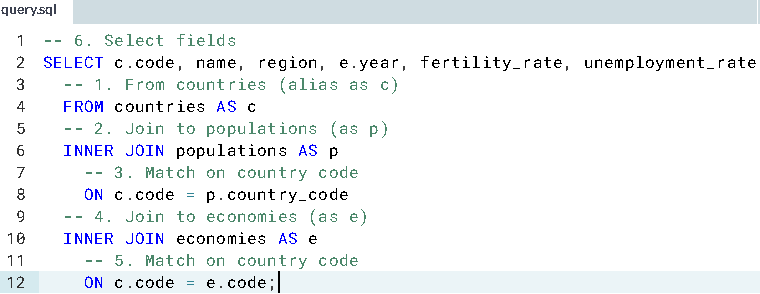
Ex.

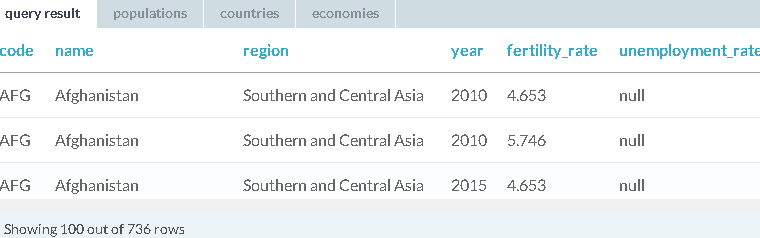
Passo 1: INNER JOIN countries (esquerda) e populations (direita) nos campos de code e country\_ code, respectivamente. - Alias countries AS c e population AS p. - Selecione o code, name e region de countries e também selecione o year e o fertility\_rate de populations (5 campos no total).





Passo 2: Adicione um INNER JOIN adicional com economies à sua consulta anterior juntando no código. - Inclua a coluna unemployment\_rate que ficou disponível por meio da junção com economies. - Observe que year aparece em populations e economies; portanto, você deve usar explicitamente “e.year” em vez de ano.

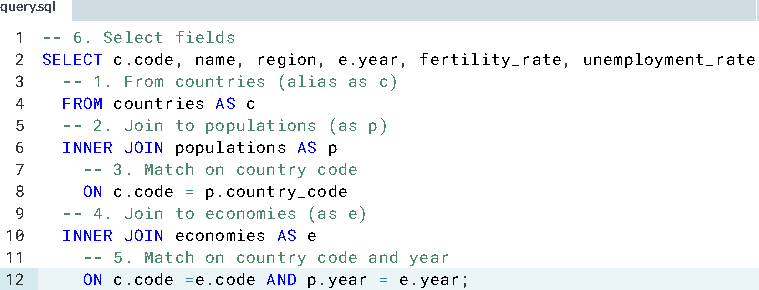


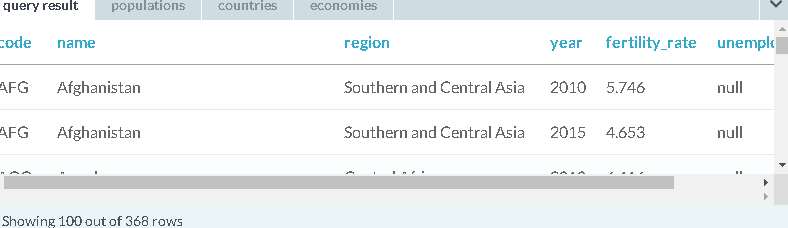


**Passo 3**: Role para baixo o resultado da consulta e dê uma olhada nos resultados da Albânia da sua consulta anterior. Algo parece estranho para você?

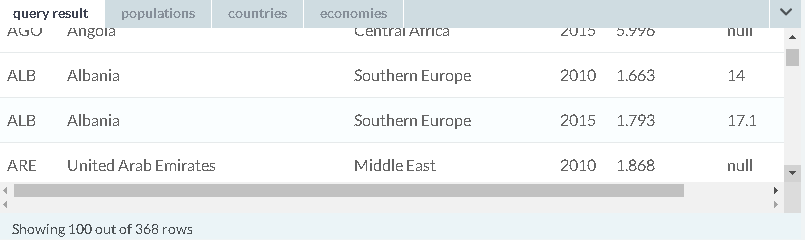
O problema de fazer sua última associação em c.code = e.code e não incluir também o ano é que, por exemplo, o valor de 2010 para fertility\_rate também é emparelhado com o valor de 2015 para unemployment\_rate.

Corrija sua consulta anterior: na sua última cláusula ON, use AND para adicionar uma condição de associação. Além de ingressar no código em c e e, também junte no year em e e p.



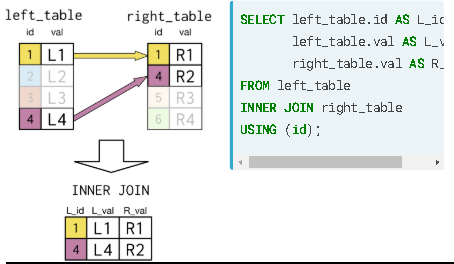


Veja que agora Albania tem duas linhas, antes tinhamos 4:

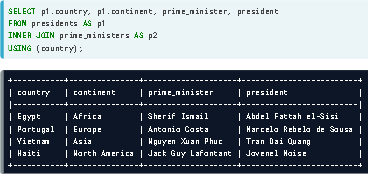


**INNER JOIN via USING**

Quando o campo-chave que você quer juntar tem o mesmo nome nas duas tabelas podemos usar a cláusula USING ao invés de ON. O campo “id” tem o mesmo nome em ambas as tabelas da esquerda e da direita, assim podemos especificar USING ao invés de ON. O parênteses é necessário ao redor do nome do campo

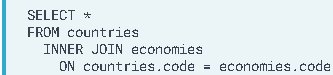


Vamos relembrar a tabela dos presidents e prime\_ministers para determinar quais países tem os dois líderes.

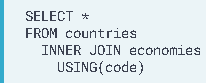


**REVIEW INNER JOIN USING ON**

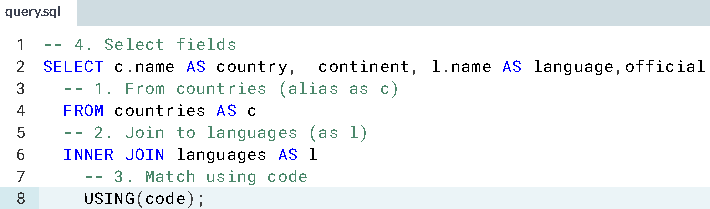
Ao unir tabelas com um nome de campo comum, por ex.:

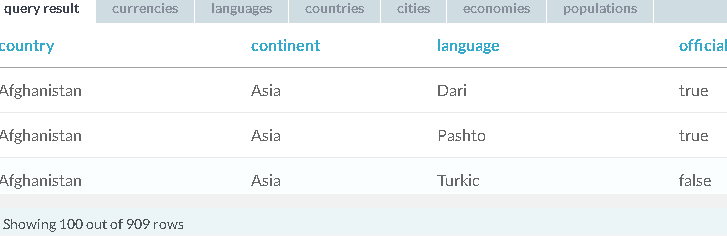


Podemos usar USING como atalho:



Ex. - INNER JOIN (associe) countries à esquerda e os languages à direita com USING (code). - Selecione os campos correspondentes a: nome do país AS country, nome do continente, nome do idioma AS language, e se o idioma é oficial ou não. Lembre-se de alias suas tabelas usando a primeira letra de seus nomes.

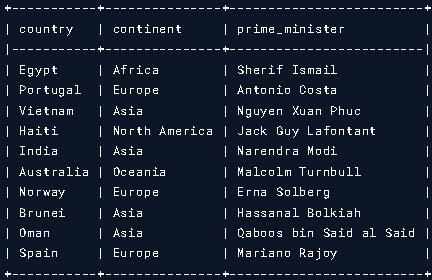




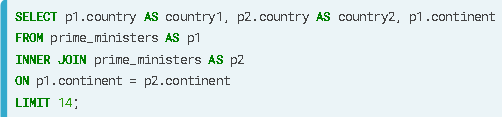
**Self Joins – (Self-ish joins) , just in CASE**

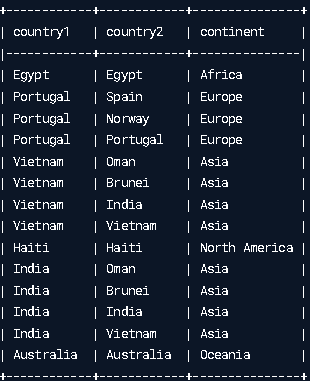
Vamos nos aprofundar em INNER JOINs em que a tabela é juntada com ela mesma (egoísta). Esse tipo de JOINs são chamos de Self Joins. Vamos aprender como fatiar um campo numérico em categorias usando o comando CASE. Juntar uma tabela nela mesma parece meio estranho.

As auto-junções são usadas para comparar valores em um campo com outros valores do mesmo campo de dentro da mesma tabela. Ex. da tabela prime\_minister:



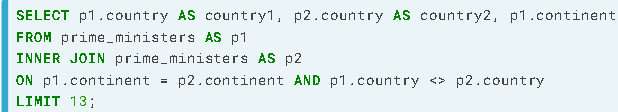
Se você quiser criar uma nova tabela mostrando countries que estão no mesmo continente combinados como pares? Veja que a coluna country e continent é selecionada duas vezes . A tabela prime\_minister está à esquerda e à direita. O passo vital aqui é estabelecer as colunas-chave pela qual nós combinaremos a tabela a ela msma. Para cada país teremos uma combinação se o país da tabela direita (q tbm é prime\_ministers) for no mesmo continente. Por fim, já que os resultados desta consulta são maiores do que pode caber no slide da aula, veremos apenas os primeiros 14 registros.





Veja que temos exatamente isso no resultado. É um emparelhamento de cada país com cada país no mesmo continente. Mas há um problema aqui. Nós não queremos listar os países com eles mesmos. Nós não queremos incluir linhas onde os países são os mesmos nos campos contry1 e country2.

A cláusula AND pode checar multiplas condições. Aqui uma combinação não será feita entre prime\_ministers e ela mesma se os países combinarem.

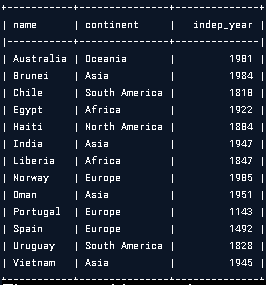


Agora teremos a tabela correta. Os resultados foram limitados para caber no slide da aula. Observe que o self-join não tem uma sintaxe tão simples quanto INNER JOIN

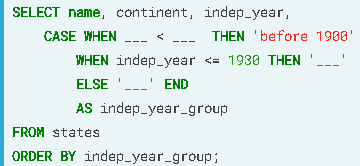


O próximo comando não é JOIN, mas é uma ferramenta útil no nosso repertório: CASE WHEN and THEN.

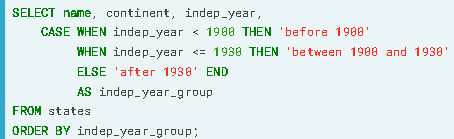
Vamos usar CASE com outra tabela da base de dados do líderes. A tabela States contém dados numéricos sobre diferentes países nos 6 continentes. Suponha que eu queira agrupar o ano da independencia em categorias de antes de 1900, entre 1900 e 1930 e depois de 1930. Vamos usar CASE, que é uma forma de fazer múltiplas declarações if-then-else (se-se não- então) de forma simplificada no SQL.

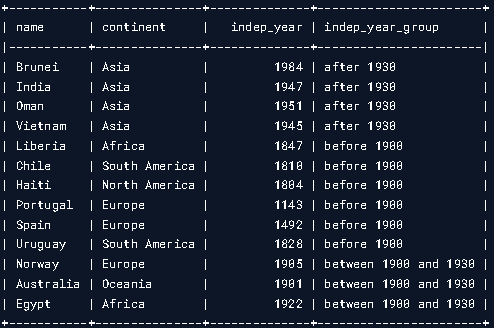


Veja o layout básico para criar um novo campo contendo os agrupamentos.



Para preenchê-lo, depois do 1º WHEN devemos especificar q queremos checar indep\_year sendo menor que 1900. Em seguida (no 2º branco), queremos que o indep\_year contenha “entre 1900 e 1930”. Por fim registros que não combinem com essas condições serão atribuidos ao valor “after 1930” no indep\_year\_group. Veja aqui a query preenchida:





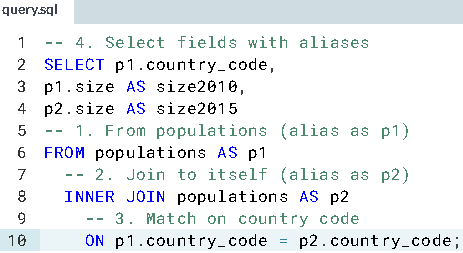
Note tbm como os continentes se relacioname com as datas de independencia.

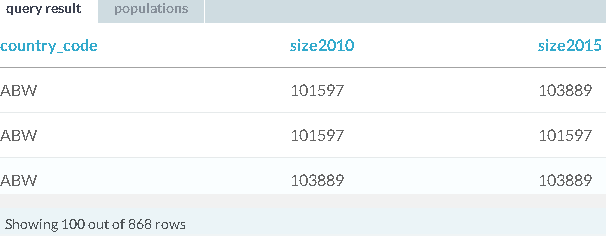
**Ex. Self-join: Self-join:**

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.

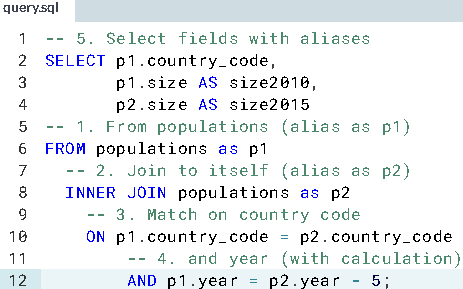
Passo 1 -Join populations with itself ON country\_code. - Select the country\_code from p1 and the size field from both p1 and p2. SQL won't allow same-named fields, so alias p1.size as size2010 and p2.size as size2015.





Passo 2: 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.

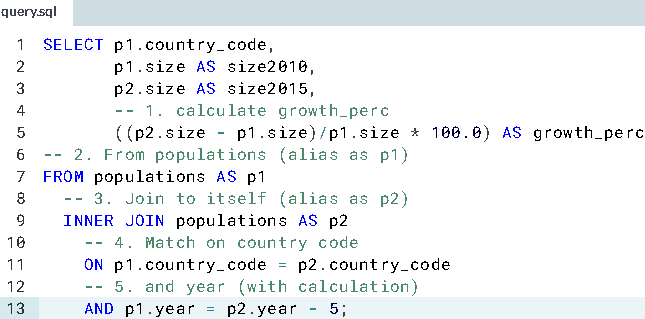


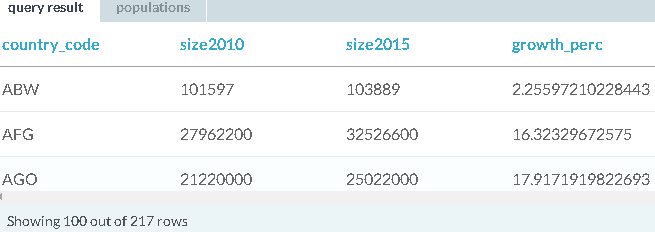


**Passo 3:** As you just saw, you can also use SQL to calculate values like p2.year - 5 for you. With two fields like size2010 and size2015, you may want to determine the percentage increase from one field to the next:

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.





**Ex. Case when and then**

Freqüentemente, é útil olhar para um campo numérico não como dados brutos, mas como estando em diferentes categorias ou grupos.

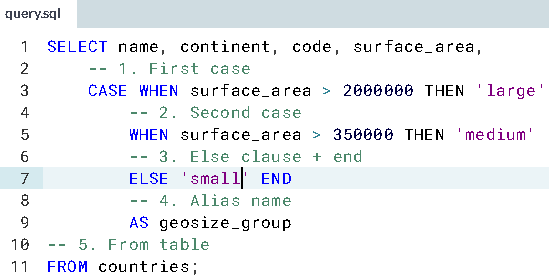
Você pode usar CASE com WHEN, THEN, ELSE e END para definir um novo campo de agrupamento.

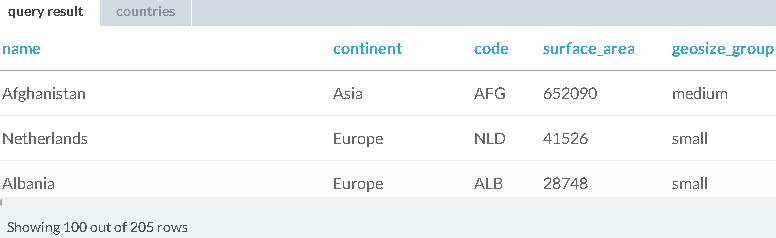
Ex. Using the countries table, create a new field AS geosize\_group that groups the countries into three groups:

- 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'.

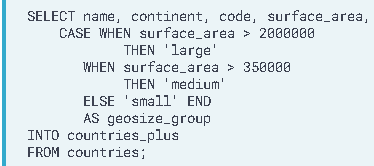
- Otherwise, geosize\_group is 'small'.

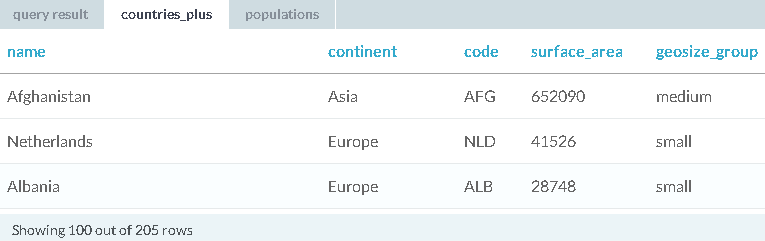




**Ex. Desafio:**

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 [a consulta que fiz no exercício anterior foi salva como tabela usando o INTO]:





Agora você explorará a relação entre o tamanho de um país em termos de área de superfície e em termos de população usando campos de agrupamento criados com o CASE.

Ao final deste exercício, você escreverá duas consultas consecutivas em um único script.

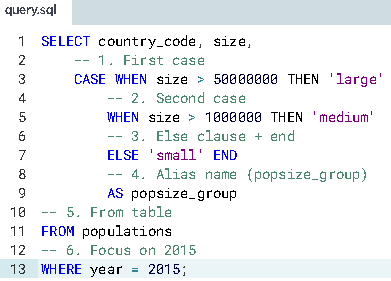
**Passo 1: Using the populations table focused only for the year 2015, create a new field aliased as popsize\_group to organize population size into**

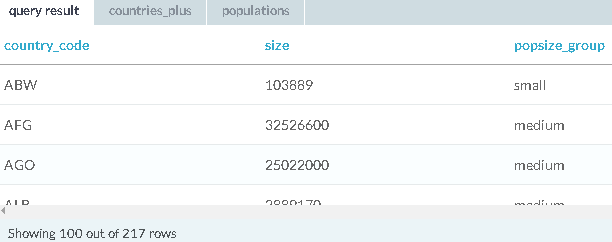
**- 'large' (> 50 million),**

**- 'medium' (> 1 million), and**

**- 'small' groups.**

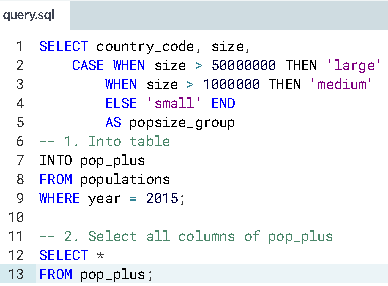
**Select only the country code, population size, and this new popsize\_group as fields.**

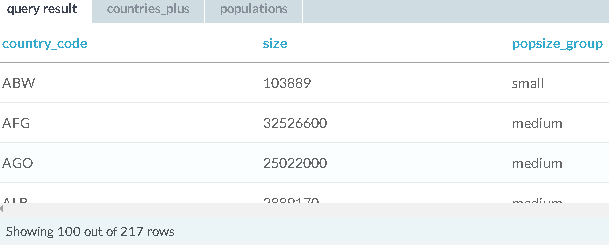




**Passo 2: Use INTO to save the result of the previous query as pop\_plus. You can see an example of this in the countries\_plus code in the assignment text. Make sure to include a ; at the end of your WHERE clause!**

**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.**



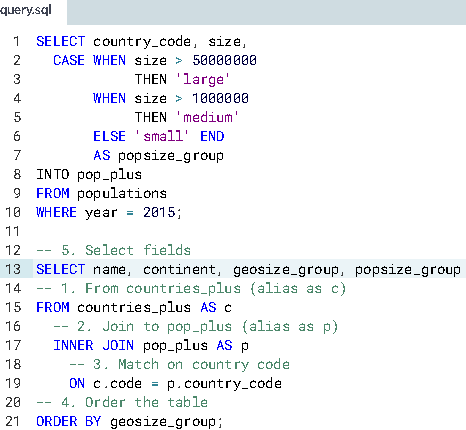


**Passo 3: - Keep the first query intact that creates pop\_plus using INTO.**

**- 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.**



**LEFT and RIGHT JOINs (ou outer joins – full, right, left joins)**

As junções externas são como estender a mão para outra tabela enquanto mantém todos os registros da tabela original. Os INNER JOINs (junções internas) mantêm apenas os registros nas duas tabelas.

P.S.: **OUTER** vai te dar todas as linhas da tabela A e todas as linhas da tabela B, mesmo que não haja correspondência.

**INNER** vai te dar apenas as tuplas de ambas as tabelas que tem correspondência da chave que escolher para sua junção.

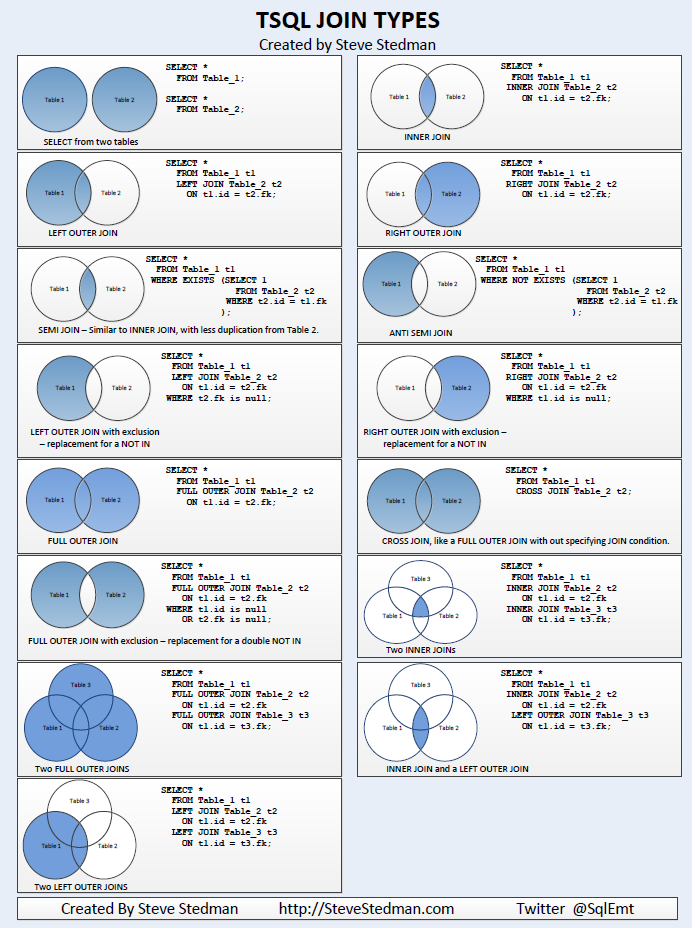
**Ex:**

a tabela A tem como chave da junção os valores: 1 | 2 | 3

E a tabela B: 2|3|4

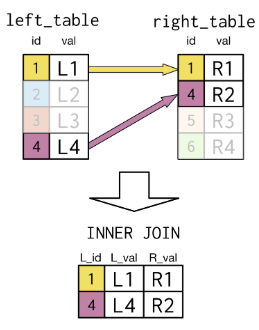
Um OUTER JOIN por essa chave entre essas tabelas resultaria em: 1|2|3|4

E um INNER JOIN resultaria em apenas: 2 | 3

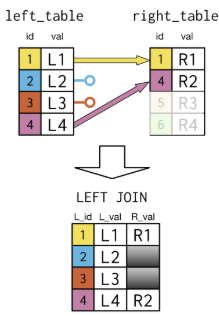


Há 3 tipos de Outer joins: right, left, full.

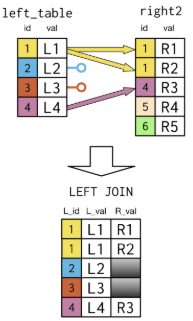
Diferença do INNER JOIN e LEFT JOIN: no INNER JOIN apenas os registros do campo id que se encontram em ambas as tabelas aparecem nos reusltados.



Já no LEFT JOIN, veja que há registros da tabela esquerda que não possuem valores correspondentes no campo da tabela direita (os campos de id 2 e 3 da tabela esquerda não tem correspondente na tabela direira). O resultado de um query usando LEFT JOIN é que ele mantém todos os registros originais da tabela esquerda, mas marca os valores que faltaram na tabela direita, pois não tem nenhum correspondente desse resgirto da tabela esquerda na tabela direita. Veja que os valores de id 5 e 6 da tabela direita não vão para os resultados do LEFT JOIN.

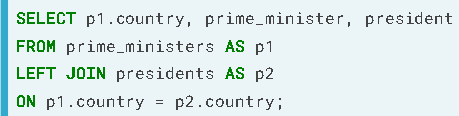


Não é sempre que cada valor-chave da tabela esquerda corresponde exatamente a um registro na coluna-chave da tabela direita.Veja esse ex.:



Há entradas faltando nos ids 2 e 3. Linhas duplicadas são mostradas no LEFT JOIN para o id 1, já q ele tem duas correspondências na tabela direita.

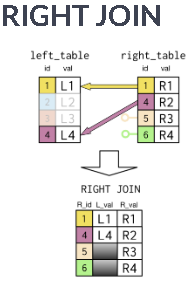
A sintaxe do LEFT JOIN é similar ao INNER JOIN. Vamos explorar o mesmo código que usamos antes para determinar os countries com prime\_minister e president. Mas desta vez vamos usar LEFT JOIN ao invés de INNER JOIN e remover continent para economizar espaço na tela.



Os primeiros 4 registros nesta tabela são os mesmos do INNER JOIN. Os últimos 6 correspondem aos países que não tem um presidente, por isso seus valores estão faltando.

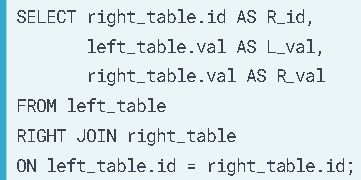


RIGHT JOIN é menos comum do que o LEFT JOIN, então vamos ver sobre ele brevemente.



Vamos ver como ele funciona. Ao invés de combinar entradas da coluna id da tabela esquerda para a tabela direita, o RIGHT JOIN faz o inverso. Vejo que na tabela direita os ids 5 e 6 não encontram correspondente na tabela esquerda. A tabela resultante do RIGHT JOIN mostra as entradas ausentes no campo L\_val.

Veja que a tabela direita aparece depois do RIGHT JOIN e tabela esquerda aparece depois do FROM.

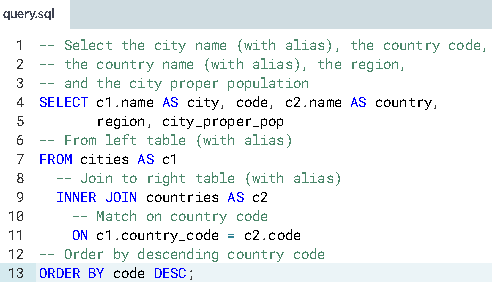


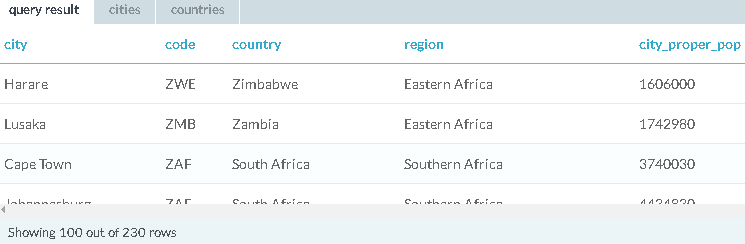
**EX LEFT JOIN:**

Vamos ver as diferenças entre usar INNER JOIN e LEFT JOIN usando as tabelas cities e countries.

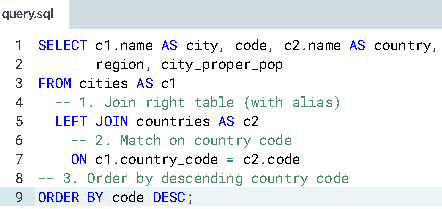
You'll begin by performing an inner join with the cities table on the left and the countries table on the right. Remember to alias the name of the city field as city and the name of the country field as country.

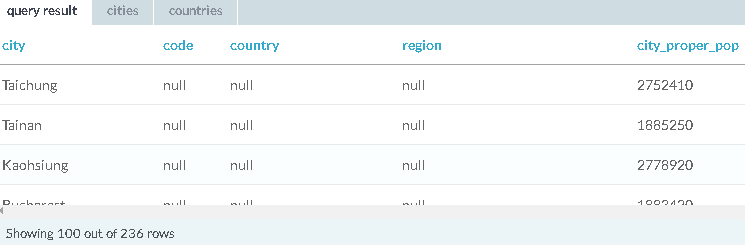
**1. Fill in the code based on the instructions in the code comments to complete the inner join. Note how many records are in the result of the join in the query result tab.**





**2.** **Change the code to perform a LEFT JOIN instead of an INNER JOIN. After executing this query, note how many records the query result contains.**





**EX. LEFT JOIN (2)**

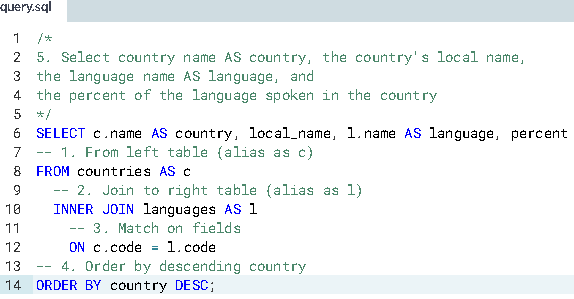
**Outro exemplo comparando INNER JOIN ao LEFT JOIN. Veja que a tabela languages possui 955 linhas e a tabela countries 205 linhas.**

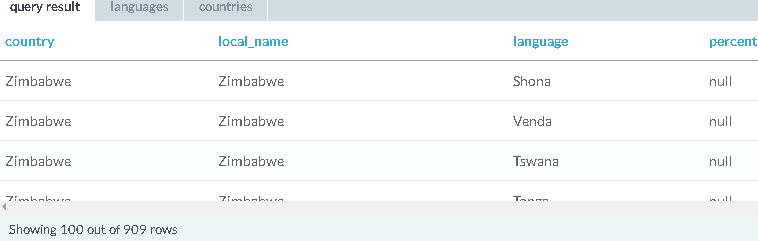
**You will begin with an inner join on the countries table on the left with the languages table on the right. Then you'll change the code to a left join in the next bullet.**

**Note the use of multi-line comments here using /\* and \*/.**

**1.** **Perform an inner join. Alias the name of the country field as country and the name of the language field as language.**

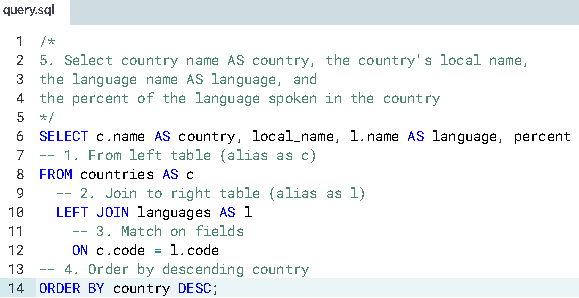
**Sort based on descending country name.**

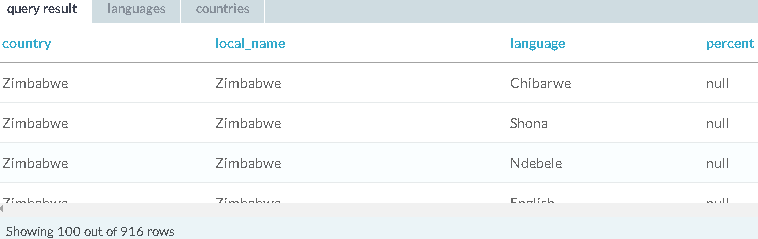




**2.** **Perform a left join instead of an inner join. Observe the result, and also note the change in the number of records in the result.**

**Carefully review which records appear in the left join result, but not in the inner join result.**

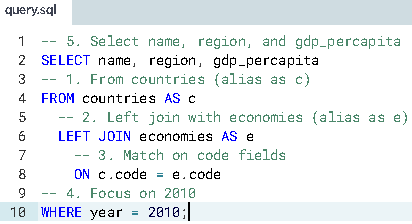


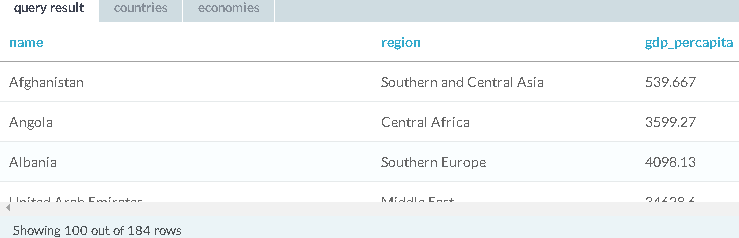


**Ex. LEFT JOIN (3)**

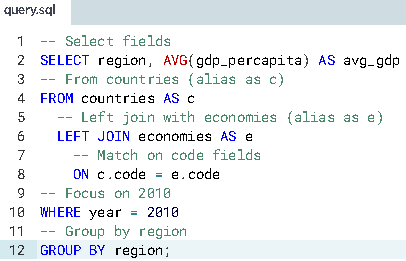
**You'll now revisit the use of the AVG() function introduced in our Intro to SQL for Data Science course. You will use it in combination with left join to determine the average gross domestic product (GDP) per capita by region in 2010.**

**Passo 1: Begin with a left join with the countries table on the left and the economies table on the right. Focus only on records with 2010 as the year.**



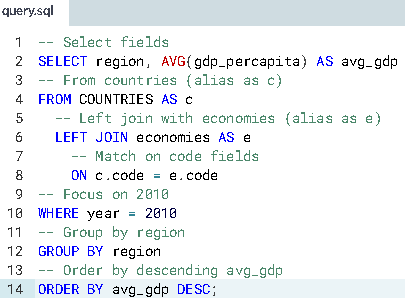


**Passo 2: Modify your code to calculate the average GDP per capita AS avg\_gdp for each region in 2010. Select the region and avg\_gdp fields.**





**Passo 3: Arrange this data on average GDP per capita for each region in 2010 from highest to lowest average GDP per capita.**

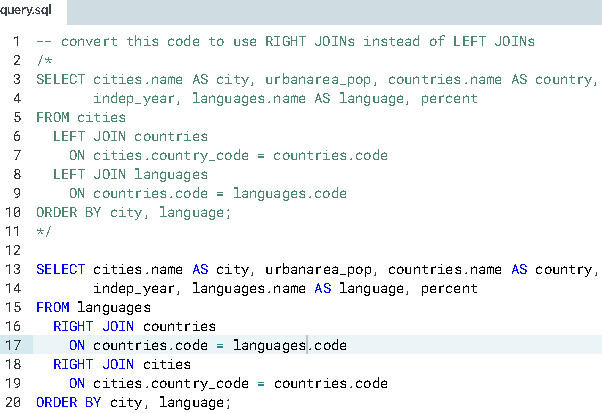


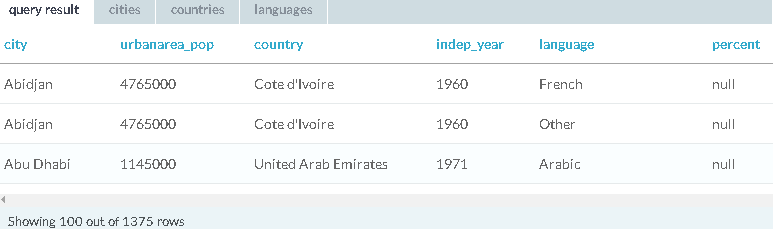


**Ex. RIGHT JOIN**

Right joins aren't as common as left joins. One reason why is that you can always write a right join as a left join.

**Ex. The left join code is commented out here. Your task is to write a new query using rights joins that produces the same result as what the query using left joins produces. Keep this left joins code commented as you write your own query just below it using right joins to solve the problem. Note the order of the joins matters in your conversion to using right joins!**



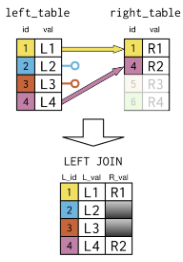
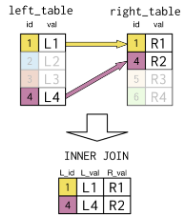


**FULL JOIN**

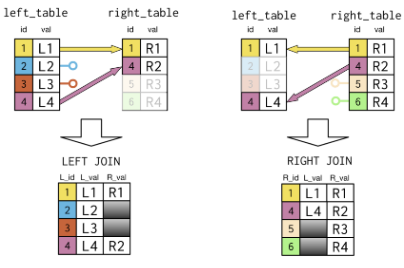
O último dos 3 tipos de OUTER JOINs é o FULL JOIN.

Vamos ver a diferença entre o FULL JOIN e os outros JOINs que aprendemos até agora (INNER JOIN, LEFT JOIN, RIGHT JOIN).

Lembrando que o INNER JOIN só mantém os registros que possuem um campo-chave com valores correspondentes nas duas tabelas. O LEFT JOIN mantém todos os regsistros da tabela esquerda e traz os valores ausentes daqueles cujos valores dos campos-chaves não aparecem da tabela direita.

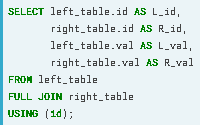
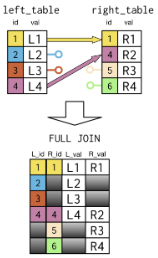


Lembrando as diferenças do RIGHT JOIN e do LEFT JOIN. Os valores id de 2 e 3 da tabela esquerda não correspondem aos valores id da tabela direita. Assim, valores ausentes são trazidos no LEFT JOIN. Do mesmo modo para RIGHT JOIN, valores ausentes são trazidos para os valores de id 5 e 6.



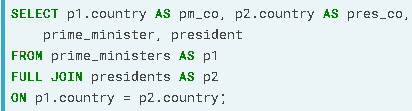
O FULL JOIN combina o LEFT JOIN e o RIGHT JOIN, veja, ele vai trazer todos os registros de ambas as tabelas esquerda e direita e rastrear os valores ausentes. Note os valores ausentes e veja que todos os 6 valores id foram incluídos na tabela.

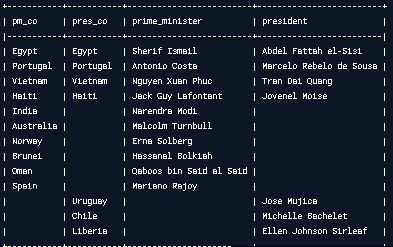
Veja também o código SQL para produzir o resultado deste FULL JOIN



Ex. da base de dados dos líderes. Os países com prime ministers e/ou presidents.

O SELECT inclui o campo COUNTRY de ambas as tabelas e os campos prime\_minister e president. O FROM especifica a tabela esquerda como prime\_ministers. A ordem importa aqui, se você mudar as duas tabelas o resultado será um pouco diferente. O FULL JOIN especifica a tabela direita. Por fim o ON diz que o JOIN é baseado no campo-chave country nas duas tabelas.





**FULL JOIN EX.**

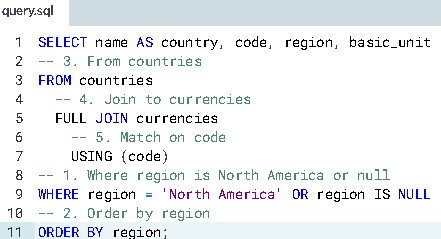
In this exercise, you'll examine how your results differ when using a full join versus using a left join versus using an inner join with the countries and currencies tables.

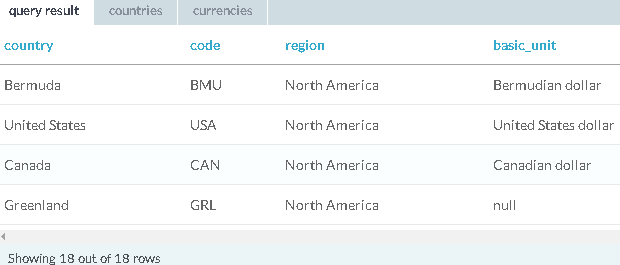
You will focus on the North American region and also where the name of the country is missing. Dig in to see what we mean!

Begin with a full join with countries on the left and currencies on the right. The fields of interest have been SELECTed for you throughout this exercise.

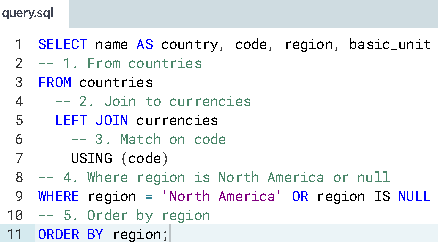
Then complete a similar left join and conclude with an inner join.

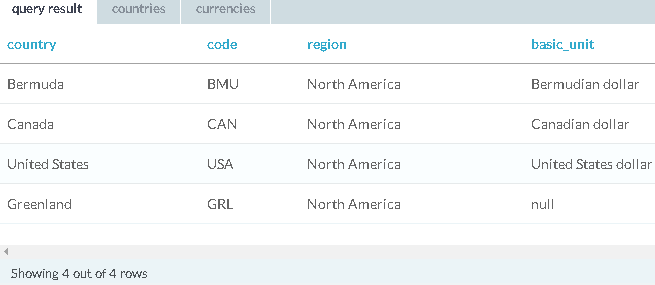
**1. Choose records in which region corresponds to North America or is NULL.**



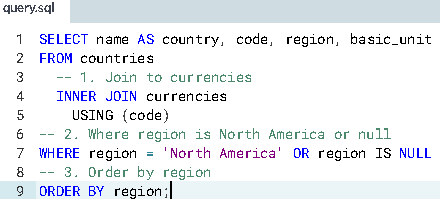


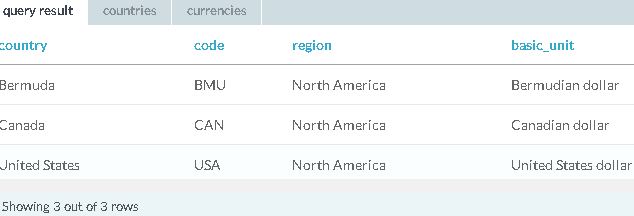
**2.** **Repeat the same query as above but use a LEFT JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN result!**





**3.** **Repeat the same query as above but use an INNER JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN and LEFT JOIN results!**





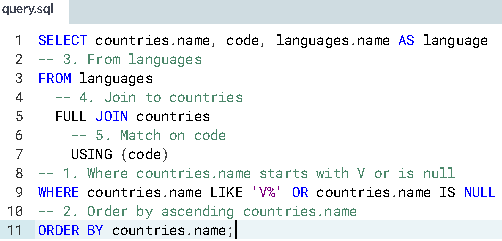
**FULL JOIN (2) EX.**

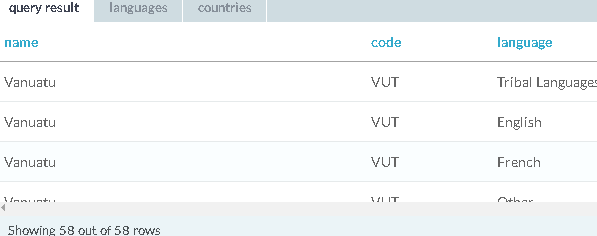
**You'll now investigate a similar exercise to the last one, but this time focused on using a table with more records on the left than the right. You'll work with the languages and countries tables.**

**Begin with a full join with languages on the left and countries on the right. Appropriate fields have been selected for you again here.**

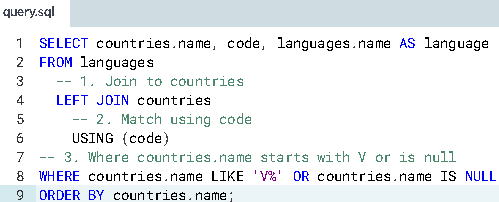
**1.** **Choose records in which countries.name starts with the capital letter 'V' or is NULL.**

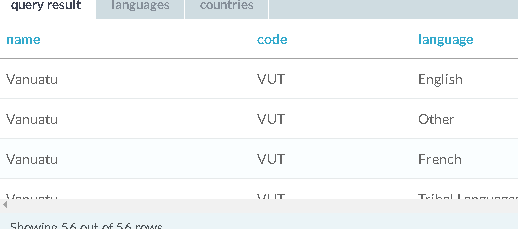
**Arrange by countries.name in ascending order to more clearly see the results.**



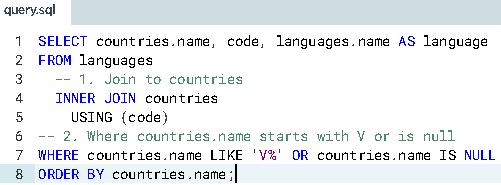


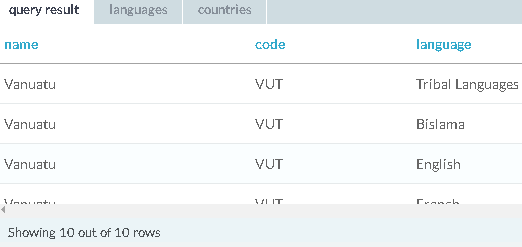
**2.** **Repeat the same query as above but use a left join instead of a full join. Note what has changed compared to the full join result!**





**3.** **Repeat once more, but use an inner join instead of a left join. Note what has changed compared to the full join and left join results.**





**FULL JOIN (3) EX**

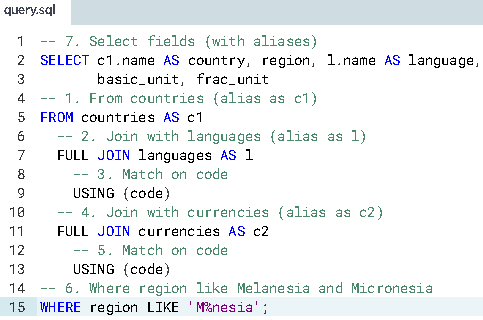
**You'll now explore using two consecutive full joins on the three tables you worked with in the previous two exercises.**

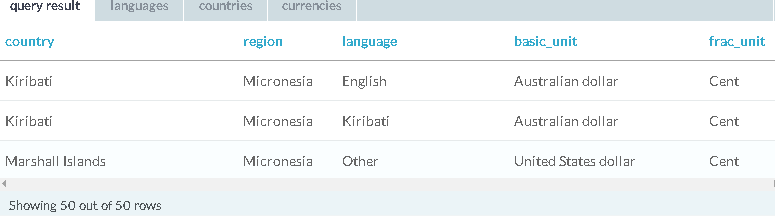
**- Complete a full join with countries on the left and languages on the right.**

**- Next, full join this result with currencies on the right.**

**- Use LIKE to choose the Melanesia and Micronesia regions (Hint: 'M%esia').**

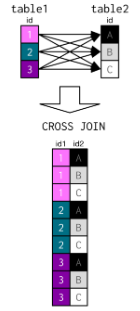
**- Select the fields corresponding to the country name AS country, region, language name AS language, and basic and fractional units of currency.**



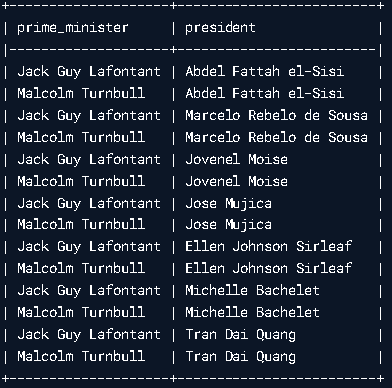
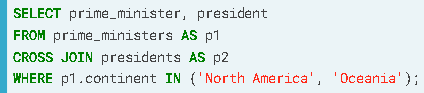


**CROSSing the Rubicon**

Agora que já vimos INNER JOINs e OUTER JOINs é hora de aprender CROSS JOIN. CROSS JOINs criam todas as combinasções possíveis de duas tabelas. Veja no diagrama temos as tabelas 1 e 2. Cada tabela tem apenas um campo de nome id. O resultado do CROSS JOIN são 9 combinações dos valores dos ids 1, 2 e 3 na tabela 1 com os valores de id A, B e C da tabela 2.



Vamos ver um ex da base de dados dos líderes. O WHERE foca só nos prime\_ministers de North America e Oceania. O resultado da query nos dá o emparelhamento dos dois prime\_ministers de North America e Oceania da tabela prime\_ministers com os 7 presidentes da tabela presidents.



**Ex. A table of two cities**

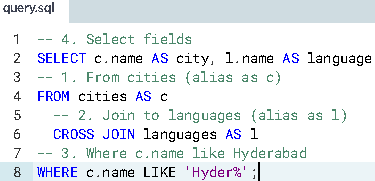
**Passo 1: This exercise looks to explore languages potentially and most frequently spoken in the cities of Hyderabad, India and Hyderabad, Pakistan.**

**You will begin with a cross join with cities AS c on the left and languages AS l on the right. Then you will modify the query using an inner join in the next tab.**

**- Create the cross join as described above. (Recall that cross joins do not use ON or USING.)**

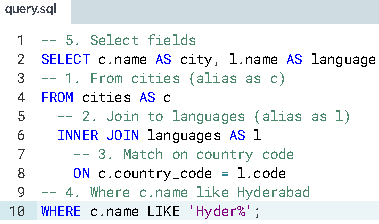
**- Make use of LIKE and Hyder% to choose Hyderabad in both countries.**

**- Select only the city name AS city and language name AS language.**





**Passo 2: Use an inner join instead of a cross join. Think about what the difference will be in the results for this inner join result and the one for the cross join.**



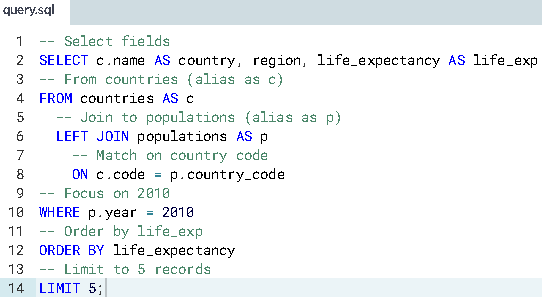


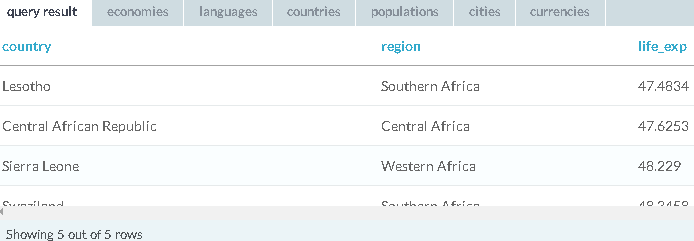
**EX. OUTER CHALLENGE**

**In terms of life expectancy for 2010, determine the names of the lowest five countries and their regions.**

**- Select country name AS country, region, and life expectancy AS life\_exp.**

**- Make sure to use LEFT JOIN, WHERE, ORDER BY, and LIMIT.**





**Cáp. 3 – Set Theory clauses**

**State of the UNION**

Veremos “Set theory clauses – Cláusulas da teoria dos conjuntos”. Usaremos o digrama Venn para representar estas operações.

Cada círculo no diagram Venn representa uma tabela. O sombreado representa o que é incluído no resultado da operação dos conjuntos em cada tabela.

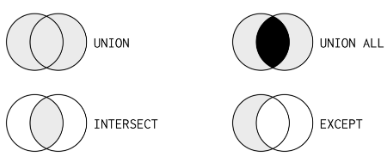
A **união** inclui todos os registros de ambas as tabelas, mas não conta duas vezes aqueles que estão em ambas as tabelas.

**UNION ALL** inclui todos os registros de ambas as tabelas e replica aqueles que estão em ambas tabelas. Por isso o centro está em preto.

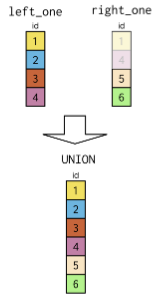
Os dois diagramas abaixo representam apenas os subconjuntos de dados sendo selecionados.

**INTERSECT** resulta em apenas aqueles registros encontrados nas duas tabelas.

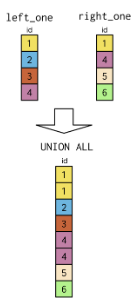
**EXCEPT** resulta apenas nos registros de uma tabela, mas não na outra.



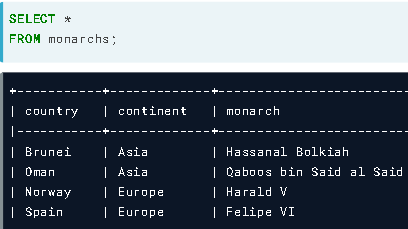
Veja como UNION se parece em um diagrama de join. Neste diagrama, você tem duas tabelas left e right com apenas um campo. Se rodarmos uma união desses dois campos, vamos obter o result o resultado abaixo, cada registro aparecendo em cada tabela será mostrado, mas se o id da tabela left aparece na right, esse id não é duplicado no resultado da união.



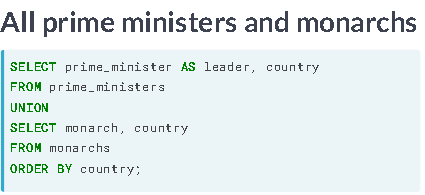
Em contrapartida, se fizermos UNION ALL das duas tabelas, o resultado incluirá os ids duplicados. Cada tabela tem 4 registros, mas como resultado de union all ficam com 8 regsitros.



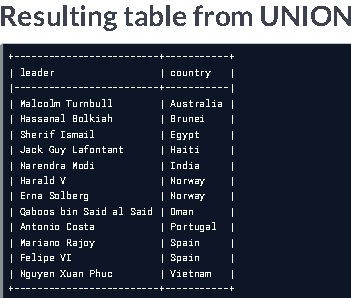
Veja o ex da base de dados dos líderes. Veja a monarchs table da base de dados dos líderes.



Podemos usar Union na tabela dos prime ministers e monarchs para mostrar todos os diferentes primeiro ministros e monarcas das duas tabelas. Os campos incluídos na operação devem ser do mesmo tipo de dado, já que eles retornam como um só campo. Não dá para empilhar um número no topo do campo do caracter.



A tabela resultante de UNION mostra todos os lpideres e seus países correspondentes.



Veja que os países oman e Brunei só foram listados uma vez na tabela de union. Esses países têm monarcas que tbm agem como prime ministers. Isso pode ser visto nos resultados de UNION ALL.

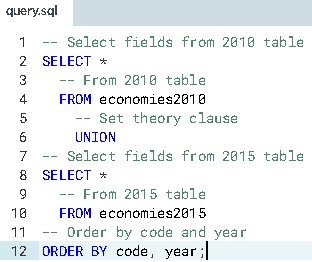


**EX. UNION**

Near query result to the right (aba da tela em que o resultado da minha query aparecerá), you will see two new tables with names economies2010 and economies2015.

- Combine these two tables into one table containing all of the fields in economies2010. The economies table is also included for reference.

- Sort this resulting single table by country code and then by year, both in ascending order.

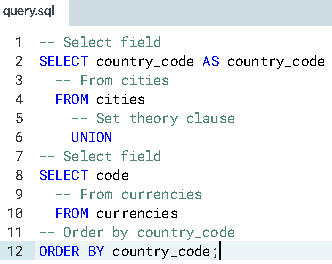
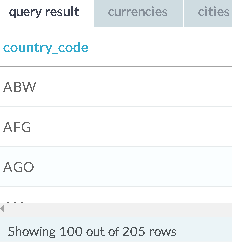


**EX. UNION (2)**

UNION can also be used to determine all occurrences of a field across multiple tables. Try out this exercise with no starter code.

- Determine all (non-duplicated) country codes in either the cities or the currencies table. The result should be a table with only one field called country\_code.

- Sort by country\_code in alphabetical order.

**EX. UNION ALL**

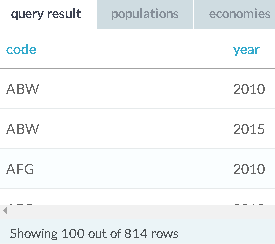
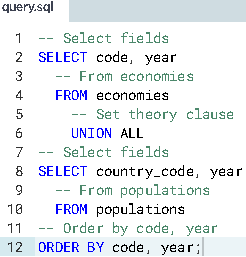
As you saw, duplicates were removed from the previous two exercises by using UNION.

To include duplicates, you can use UNION ALL.

- Determine all combinations (include duplicates) of country code and year that exist in either the economies or the populations tables. Order by code then year.

- The result of the query should only have two columns/fields. Think about how many records this query should result in.

- You'll use code very similar to this in your next exercise after the video. Make note of this code after completing it.

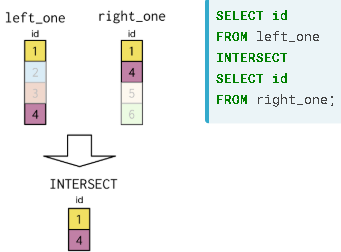


**INTERSECTional data Science**

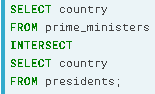
Você viu que UNION e UNION ALL (unem dois SELECTs) não fazem exatamente a mesma coisa que um JOIN faz. Eles apenas vinculam campos uns sobre os outros nas duas tabelas.

A claúsula INTERSECT da teoria dos conjuntos funciona de maneira similar a UNION e UNION ALL, mas lembre-se do diagrama Venn que INTERSECT apenas inclui registros em comum com ambas as tabelas e campos selecionados.

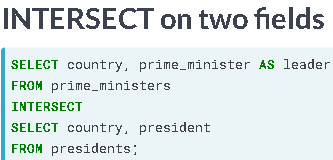
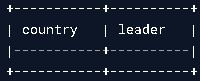
Veja o diagrama do INTERSECT e seu respectivo código. O resultado d query são apenas os resultados em comum com ambas as tabelas, ou seja, 1 e 4.



Ex. INTERSECT para determinar todos os países que possuem um prime minister e president. O código para cada um desses conjuntos de operações tem um layout similar. Veja que primeiro SELECT cada campo que você quer incluir na sua primeira tabelas e pois você especifica o nome da primeira tabela. Segundo, você especifica o nome da operação de conjunto e por fim indica quais campos da segunda tabela você quer incluir e depois o nomes da segunda tabela. O resultado da query são os 4 países que tem tanto prime minister e president na base de dados leaders.

Se tentarmos SELECT duas colunas ao invés de uma nos nossos exemplos anteriores? Será que esse tipo de seleção te devolveria como resultado o nome dos países que tem prime minister e president. Não, o resultado seria uma tabela vazia. Isso porque quando o INTERSECT procura em duas colunas ele inclui ambas colunas na busca. Assim ele não encontra nenhum país com primeiro ministro e presidente com o mesmo nome. O INTERSECT procura por registros em comum, não por campos-chave individuais como o join faz para corresponder. Essa é uma distinção importante.

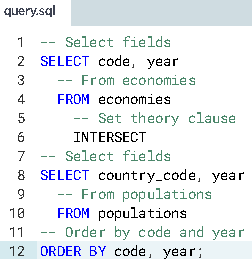
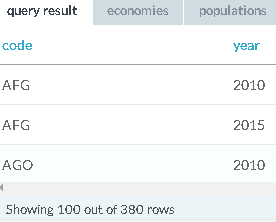
 

**EX. INTERSECT**

Repeat the previous UNION ALL exercise, this time looking at the records in common for country code and year for the economies and populations tables.

- Again, order by code and then by year, both in ascending order.

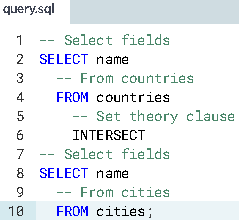
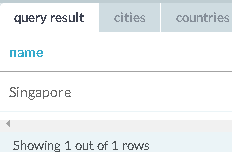
- Note the number of records here (given at the bottom of query result) compared to the similar UNION ALL query result (814 records).

**Ex. INTERSECT (2)**

As you think about major world cities and their corresponding country, you may ask which countries also have a city with the same name as their country name?

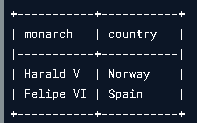
- Use INTERSECT to answer this question with countries and cities!

**EXCEPTional**

A claúsula EXCEPT permite que você inclua apenas os resgistros que estão em uma tabela, mas não na outra. Ex da base de dados dos leaders, sabemos que alguns monarcas tbm são primeiro ministros em seus países. Uma maneira de determinar essas monarcas na tabela monarchs que não tem título de primeiro ministro é usar a claúsula EXCEPT.

Essa query seleciona o campo monarch da tabelas monarchs e depois olha para entradas comuns no campo prime\_ministers, enquanto rastreia o country de cada leader. Veja que o resultado dessa query apenas dois european monarchs não são também prime ministers na base de daods dos leaders.

Este diagrama fornece a estrutura das claúsulas EXCEPT. Apenas os regsitros que aparecem na tabela left\_one, mas que não aparecem na tabela right\_one são incluidos.

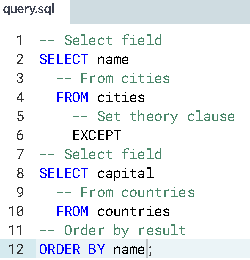
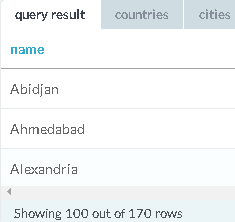
**EX. EXCEPT**

Get the names of cities in cities which are not noted as capital cities in countries as a single field result.

Note that there are some countries in the world that are not included in the countries table, which will result in some cities not being labeled as capital cities when in fact they are.

- Order the resulting field in ascending order.

- Can you spot the city/cities that are actually capital cities which this query misses?

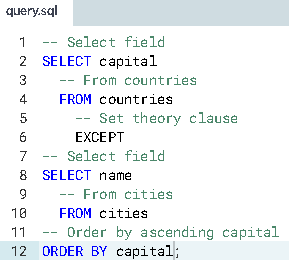
**EX. EXCEPT (2)**

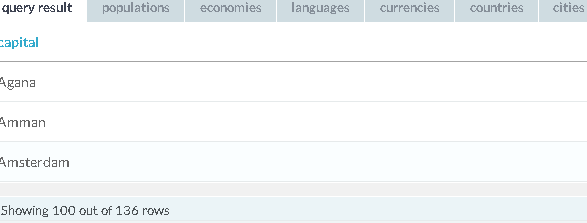
Now you will complete the previous query in reverse!

Determine the names of capital cities that are not listed in the cities table.

- Order by capital in ascending order.

- The cities table contains information about 236 of the world's most populous cities. The result of your query may surprise you in terms of the number of capital cities that DO NOT appear in this list!

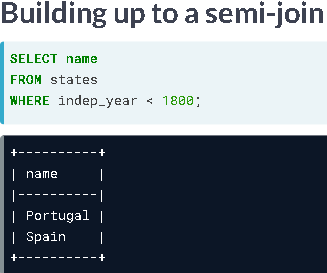




**Semi-joins and Anti-joins**

Vamos fechar o cápitulo retornando aos JOINs. Os 6 JOINs (INER JOIN, SELF JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN ) q estudamos até agora são aditivos, já q adicionam colunas a tabela original left. Os últimos dois JOINs q estudaremos usam a tabela direita para determinar quais registros manter na tabela esquerda. Ou seja, estes últimos dois JOINs (semi-join e anti-join) de modo similar a claúsula WHERE dependendo dos valores da segunda tabela.

Exemplos: suponha que vc queira determinar os presidentes dos países que ganharam independência depois de 1800. Primeiro vamos determinar quais países isso corresponde na tabela states. Antes de saber como usar JOINs isso poderia ser feito assim:

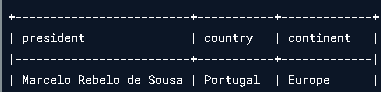


Para conseguir apenas os países que têm essa condição vc pode usar WHERE. Agora vamos estabelecer a outra parte da query para obter os presidentes que queremos. Qual o códig necesário para extrair as colunas presidente, país e continente da tabela president, nessa ordem? Agora precisamos usar esse resultado com aquele de cima para filtrar o campo país da tabela presidente e dar o resusltado correto.



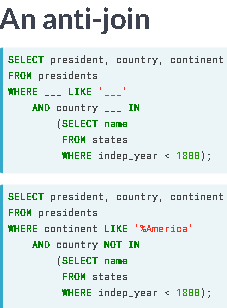
Na 1ª query desse exemplo, determinamos q Portugal e Espanha foram independentes antes de 1800. Na 2ª query, determinamos como mostrar a tabela de uma maneira legal para responder essa questão. Para combinar as duas tabelas juntas usaremos de novo WHERE e depois usar a primeira query como condição para checar a claúsula WHERE. Esse é um exemplo de subquery, uma query que está dentro de outra query [veremos mais sobre subquey no próximo cáp.]. O resultado dessa query é Portugal apenas, já que Espanha não possui presidente.

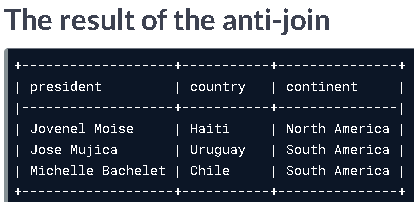




O semi-join escolhe registros da primeira tabela onde uma condição é encontrada na segunda tabela. Um anti-join escolhe registros na primeira tabela onde uma condição não é encontrada na segunda tabela.

Como você determinaria países das Americas fundados depois de 1800? Usando o código do exemplo passado, você só precisa adicionar alguns pedaços de código:

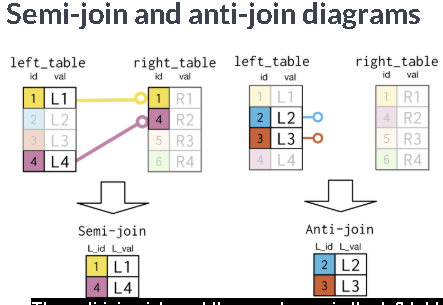




Veja que preenchemos a claúsula WHERE com os continentes que tenham como final “America”. Dpois preenchemos com NOT para excluir os países da subquery.

Os presidentes dos países na America fundados depois de 1800 estão nessa tabela de resultados.

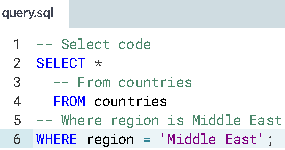
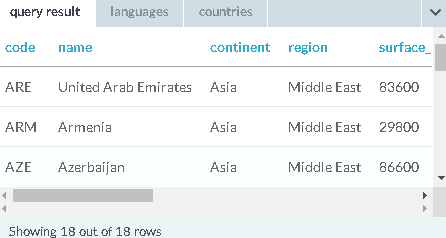
O semi-join combina registros pelo campo-chave da tabela direita com aqueles da esquerda. Dpois escolhe apenas as linhas da tabela esquerda que combinam com essa condição. O anti-join escolhe as colunas da tabela esquerda q não combinam com a condição da tebela direita. Semi-joins e anti-joins não tem a mesma sintaxe no SQL que nem o INNER JOIN e o LEFT JOIN. São ferramentas úteis para filtrar registros de tabelas nos registros de outra tabela.



**Ex. Semi-join**

You are now going to use the concept of a semi-join to identify languages spoken in the Middle East.

**Passo 1:** Flash back to our Intro to SQL for Data Science course and begin by selecting all country codes in the Middle East as a single field result using SELECT, FROM, and WHERE.

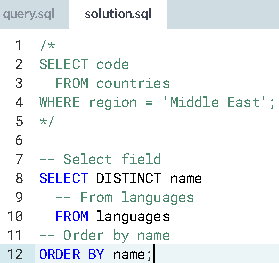
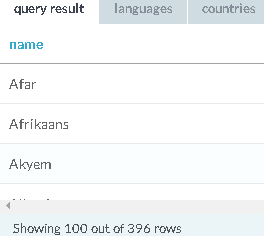
 

**Passo 2:**

**-** Comment out the answer to the previous tab by surrounding it in /\* and \*/. You'll come back to it!

- Below the commented code, select only unique languages by name appearing in the languages table.

- Order the resulting single field table by name in ascending order.

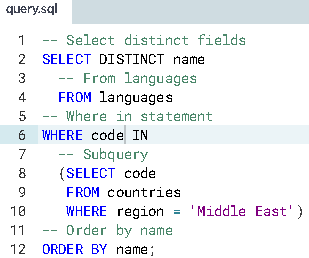
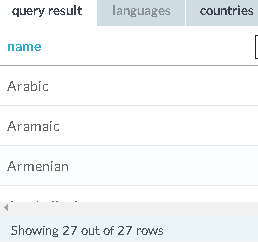
 

**Passo 3:**

Now combine the previous two queries into one query:

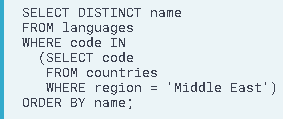
- Add a WHERE IN statement to the SELECT DISTINCT query, and use the commented out query from the first instruction in there. That way, you can determine the unique languages spoken in the Middle East.

Carefully review this result and its code after completing it. It serves as a great example of subqueries, which are the focus of Chapter 4.

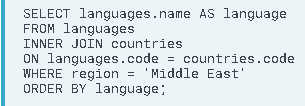
 

**Relating semi-join to a tweaked inner join - Relacionando a Semi-Junção a uma Junção Interna Ajustada**

Let's revisit the code from the previous exercise, which retrieves languages spoken in the Middle East.



Sometimes problems solved with semi-joins can also be solved using an inner join.



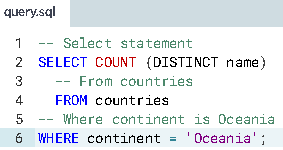
This inner join isn't quite right. What is missing from this second code block to get it to match with the correct answer produced by the first block? **DISTINCT**

**EX. Diagnosing problems using anti-join**

Another powerful join in SQL is the anti-join. It is particularly useful in identifying which records are causing an incorrect number of records to appear in join queries.

You will also see another example of a subquery here, as you saw in the first exercise on semi-joins. Your goal is to identify the currencies used in Oceanian countries!

**Passo 1:** Begin by determining the number of countries in countries that are listed in Oceania using SELECT, FROM, and WHERE.

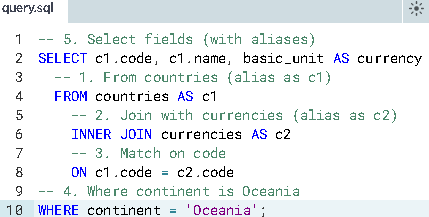
**Passo 2:**

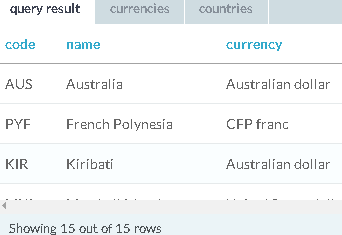
**-** Complete an inner join with countries AS c1 on the left and currencies AS c2 on the right to get the different currencies used in the countries of Oceania.

- Match ON the code field in the two tables.

- Include the country code, country name, and basic\_unit AS currency.

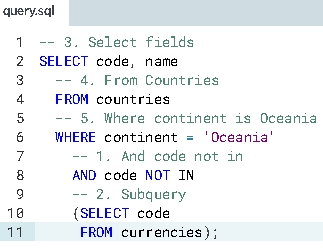
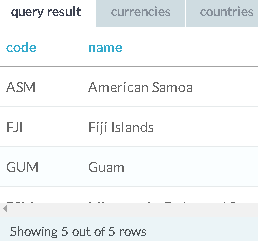
Observe query result and make note of how many different countries are listed here.





**Passo 3:** Note that not all countries in Oceania were listed in the resulting inner join with currencies. Use an anti-join to determine which countries were not included!

- Use NOT IN and (SELECT code FROM currencies) as a subquery to get the country code and country name for the Oceanian countries that are not included in the currencies table.

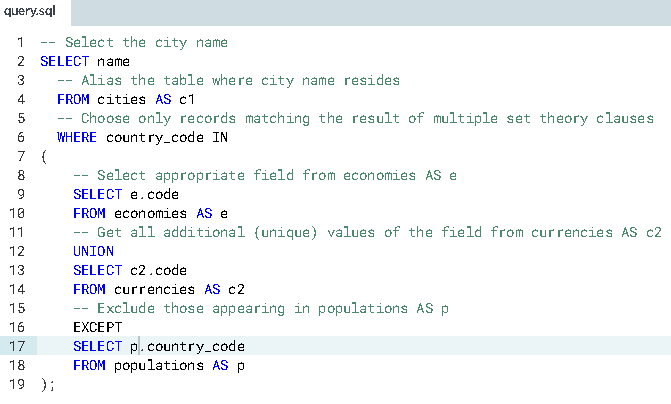
**Ex. Set theory challenge**

Your task here will be to incorporate two of UNION/UNION ALL/INTERSECT/EXCEPT to solve a challenge involving three tables.

In addition, you will use a subquery as you have in the last two exercises! This will be great practice as you hop into subqueries more in Chapter 4!

- Identify the country codes that are included in either economies or currencies but not in populations.

- Use that result to determine the names of cities in the countries that match the specification in the previous instruction.





**Cáp. 4 - Subqueries inside WHERE and SELECT clauses**

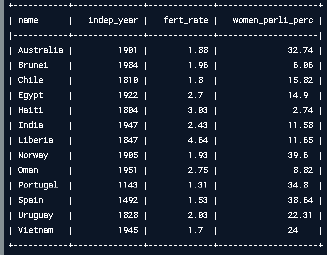
- how to use nested queries and you'll use what you’ve learned in this course to solve three challenge problems.

**4.1.Subqueries inside WHERE clause set-up**

- Cáp. sobre incorporar queries dentro de queries – queries aninhadas ou subqueries.

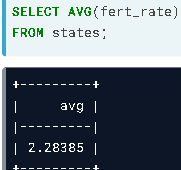
- Subquery mais comum é dentro de uma declaração WHERE;

- Ex. Tabela states. Fert\_rate estima a média do numero de bebes nascidos por mulher em cada país. O campo Women\_parli\_perc dá a porcentagem de mulheres eleitas no parlamento federal em cada país.

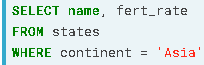


Nesses 13 países, como determinar a taxa de fertilidade média?

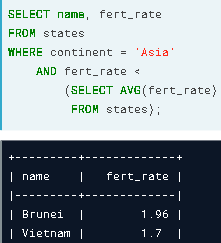
Vamos usar average fertily rate como parte da subquery. A média de bebês nascidos nesses países é de 2,28 crianças.



Vamos usar a query anterior como subquery p/ determinar os países asiáticos que se enquadram dentro dessa média. Páises asiáticos que estejam abaixo da média da subquery acima:



Veja a junção das duas consultas. O primeiro SELECT é o que vai definir os campos que aparecerão como resultado. Dentro do WHERE fazemos uma avaliação do que pode entrar no primeiro select, no caso, continentes asiaticos AND fert\_rate menor que o segundo select (q é a média da taxa de fertidalidade de todos os países da tabela. Comparamos uma média global com uma média local. Se a média local estiver abaixo da global, ela aparecerá nos resultados)



O segundo tipo mais comum de subquery inside SELECT clause. A tarefa aqui é contar o número de países listados na tabela states para cada continente na tabela prime\_ministers. Vamos passo a passo. Esse código abaixo retorna cada um dos continentes da tabela prime\_ministers.



Em seguida determinaremos os counts do número de países em states para caa continente. Combinar uma claúsula COUNT com a declaração WHERE combinando os campos continent nas duas tabelas fará isso.

A suquery envolvendo states tbm pode se referir a tabela prime\_ministers na query principal. Sempre que você quiser uma subquery dentro de uma declaração SELECT, você terá q dar a subquery um alias como countries\_num.

O primeiro select é é o campo que aparecerá nos resultados, o segundo select com alias tbm aparecerá nos resultados. Veja que na subquery igualo dois campos de tabelas diferentes.



O resultado da query acima vem abaixo. Lembre-se que sempre há várias maneiras de resolver problemas com queries no SQL.

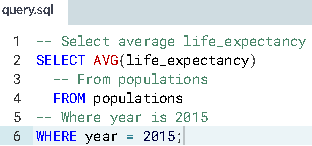
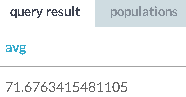


Você poderia usar um JOIN para obter esse mesmo resultado.

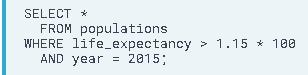
**Ex. Subquery inside where**

You'll now try to figure out which countries had high average life expectancies (at the country level) in 2015.

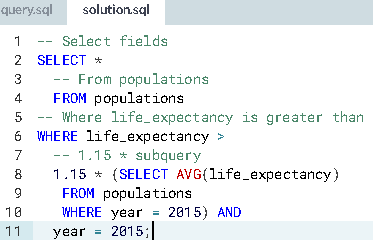
**Passo 1:** Begin by calculating the average life expectancy across all countries for 2015.

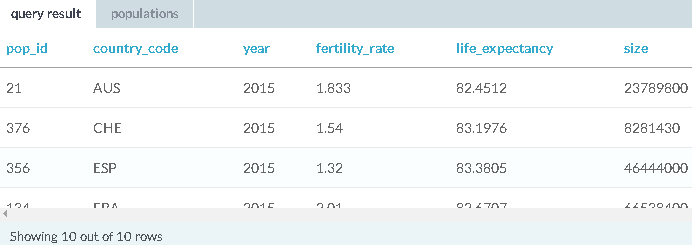
 

**Passo 2:** Recall that you can use SQL to do calculations for you. Suppose we wanted only records that were above 1.15 \* 100 in terms of life expectancy for 2015:



Select all fields from populations with records corresponding to larger than 1.15 times the average you calculated in the first task for 2015. In other words, change the 100 in the example above with a subquery.





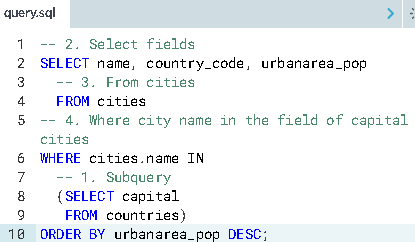
Veja que a primeira query q criamos no passo 1 é a query que está dentro do WHERE, a qual multiplicamos por 1.15, no fim ainda falamos que o ano é 2015 dentro e fora da query. Parece repetitivo, mas é que há vários anos além de 2015.

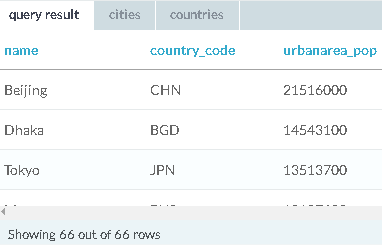
**EX. Subquery inside where (2)**

Use your knowledge of subqueries in WHERE to get the urban area population for only capital cities.

- Make use of the capital field in the countries table in your subquery.

- Select the city name, country code, and urban area population fields.





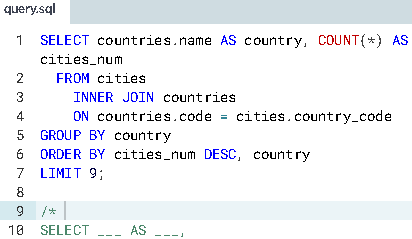
**Ex. Subquery inside select**

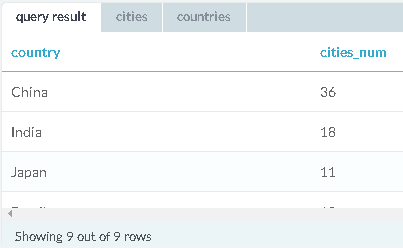
In this exercise, you'll see how some queries can be written using either a join or a subquery.

You have seen previously how to use GROUP BY with aggregate functions and an inner join to get summarized information from multiple tables.

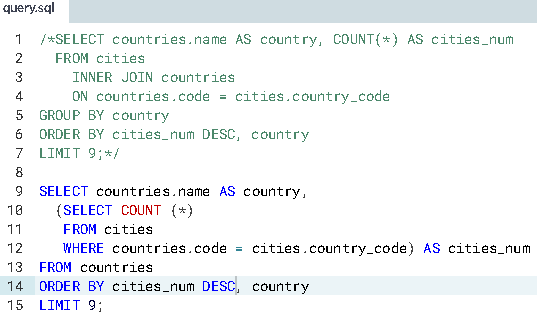
The code given in query.sql selects the top nine countries in terms of number of cities appearing in the cities table. Recall that this corresponds to the most populous cities in the world. Your task will be to convert the commented out code to get the same result as the code shown.

**Passo 1:** Just Submit Answer here!





**Passo 2:**



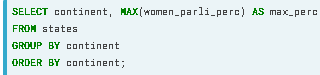


**4.2.Subquery inside the FROM clause**

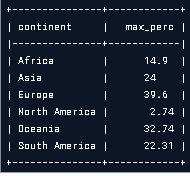
O último tipo básico de subquery é dentro da claúsula FROM.

Um exemplo motivador referente à porcentagem de mulheres no parlamento será usado agora para ajudá-lo a entender esse estilo de subquery.

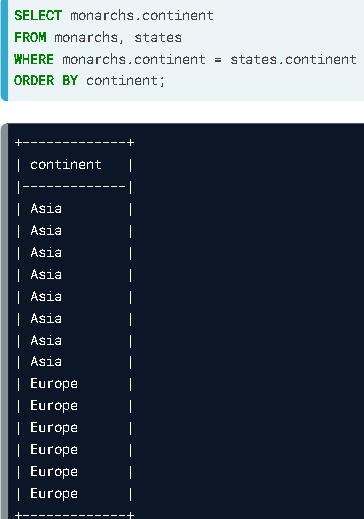
1º vamos determinar a porcentagem máxima de mulheres no parlamento para cada continente listado em states. Lembre-se que essa query apenas funcionará se vc incluir continent como um dos campos na claúsula SELECT, já que estamos agrupando baseado nesse campo.



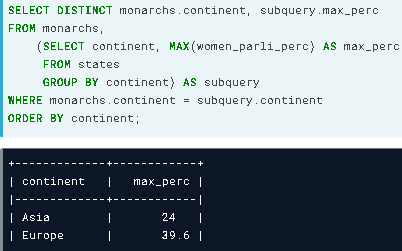
O resultado: vemos que Europa tem o maior valor e North America o menor valor dos países listaos na tabela states.



E se você não estivesse interessado em todos os continentes, mas especificamnte naqueles da tabela monarchs. Ainda não vimos isso, mas podemos incluir várias tabelas em uma claúsula FROM adicionando uma vírgula ao redor deles. Veja que temos parte da resposta, mas precisamos nos livrar das entradas duplicadas:



Para que Asia e Europa apareçam apenas uma vez, use o comando DISTINCT na declaração SELECT. Para obter a coluna máxima com Asia e Europa, ao invés de incluir states na claúsula FROM, inclua a subquery e alias ela com o nome da subquery. É assim que incluimos uma subquery como uma tabela temporária na claúsula FROM.



**Ex. Subquery inside from**

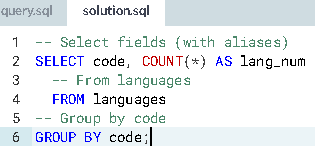
The last type of subquery you will work with is one inside of FROM.

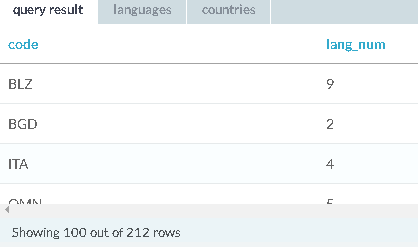
You will use this to determine the number of languages spoken for each country, identified by the country's local name! (Note this may be different than the name field and is stored in the local\_name field.)

**Passo 1:**

**-** Begin by determining for each country code how many languages are listed in the languages table using SELECT, FROM, and GROUP BY.

- Alias the aggregated field as lang\_num.





**Passo 2:**

- Include the previous query (aliased as subquery) as a subquery in the FROM clause of a new query.

- Select the local name of the country from countries.

- Also, select lang\_num from subquery.

- Make sure to use WHERE appropriately to match code in countries and in subquery.

- Sort by lang\_num in descending order.

