**Daily XP900**

**Exercise**

**Exercise**

**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!

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

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,

    -- First case

    CASE WHEN \_\_\_ > \_\_\_ THEN \_\_\_

        -- Second case

        WHEN \_\_\_ > \_\_\_ THEN \_\_\_

        -- Else clause + end

        ELSE \_\_\_ END

        -- Alias name (popsize\_group)

        AS \_\_\_

-- From table

FROM \_\_\_

-- Focus on 2015

WHERE \_\_\_ = \_\_\_;

SELECT country\_code, size,

    -- First case

    CASE WHEN size > 50000000 THEN 'large'

        -- Second case

        WHEN size > 1000000 THEN 'medium'

        -- Else clause + end

        ELSE 'small' END

        -- Alias name (popsize\_group)

        AS popsize\_group

-- From table

FROM populations

-- Focus on 2015

WHERE year = 2015;

| **country\_code** | **size** | **popsize\_group** |
| --- | --- | --- |
| ABW | 103889 | small |
| AFG | 32526562 | medium |
| AGO | 25021974 | medium |
| ALB | 2889167 | medium |
| AND | 70473 | small |
| ARE | 9156963 | medium |
| ARG | 43416756 | medium |

* 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 the query result.

SELECT country\_code, size,

    CASE WHEN size > 50000000 THEN 'large'

        WHEN size > 1000000 THEN 'medium'

        ELSE 'small' END

        AS popsize\_group

-- Into table

INTO pop\_plus

FROM populations

WHERE year = 2015;

-- Select all columns of pop\_plus

SELECT \*

FROM pop\_plus;

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;

-- Select fields

SELECT name, continent, geosize\_group, popsize\_group

-- From countries\_plus (alias as c)

FROM countries\_plus AS c

  -- Join to pop\_plus (alias as p)

  INNER JOIN pop\_plus AS p

    -- Match on country code

    ON c.code = p.country\_code

-- Order the table

ORDER BY geosize\_group;

| **name** | **continent** | **geosize\_group** | **popsize\_group** |
| --- | --- | --- | --- |
| Canada | North America | large | medium |
| United States | North America | large | large |
| Greenland | North America | large | small |
| Argentina | South America | large | medium |
| Kazakhstan | Asia | large | medium |
| China | Asia | large | large |
| Russian Federation | Europe | large | large |
| India | Asia | large | large |
| Algeria | Africa | large | medium |
| Congo, The Democratic Republic of the | Africa | large | large |
| Saudi Arabia | Asia | large | medium |
| Sudan | Africa | large | medium |
| Brazil | South America | large | large |
| Australia | Oceania | large | medium |
| Madagascar | Africa | medium | medium |
| Iraq | Asia | medium | medium |
| Mexico | North America | medium | large |
| Somalia | Africa | medium | medium |
| Botswana | Africa | medium | medium |
| Central African Republic | Africa | medium | medium |
| Mali |  |  |  |

**Daily XP1000**

# LEFT and RIGHT JOINs

**50 XP**

## 1. LEFT and RIGHT JOINs

Congratulations on completing Chapter 1 on INNER JOINs. Welcome to Chapter 2 on OUTER JOINs! You can remember outer joins as reaching OUT to another table while keeping all of the records of the original table. Inner joins keep only the records IN both tables. You'll begin this chapter by exploring (1) LEFT JOINs, (2) RIGHT JOINs, and (3) FULL JOINs, which are the three types of OUTER JOINs. Let's begin by exploring how a LEFT JOIN differs from an INNER JOIN via a diagram.

## 2. INNER JOIN diagram

Recall the inner join diagram from Chapter 1. The only records that were included in the resulting table of the INNER JOIN query were those in which the id field had matching values.

## 3. LEFT JOIN initial diagram

In contrast, a LEFT JOIN notes those records in the left table that do not have a match on the key field in the right table. This is denoted in the diagram by the open circles remaining close to the left table for id values of 2 and 3. These values of 2 and 3 do not appear in the id field of the right table.

## 4. LEFT JOIN diagram

You now see the result of the LEFT JOIN query. Whereas the INNER JOIN kept just the records corresponding to id values of 1 and 4, a LEFT JOIN keeps all of the original records in the left table but then marks the values as missing in the right table for those that don't have a match. The missing values are marked with dark gray boxes here for clarity. Note that the values of 5 and 6 for id in the right table are not found in the result of LEFT JOIN in any way.

## 5. Multiple INNER JOIN diagram

It isn't always the case that each key value in the left table corresponds to exactly one record in the key column of the right table. In these examples, we have this layout. Missing entries still occur for ids of 2 and 3 and the value of R3 is brought into the join from right2 since it matches on id 4. Duplicate rows are shown in the LEFT JOIN for id 1 since it has two matches corresponding to the values of R1 and R2 in the right2 table.

## 6. The syntax of a LEFT JOIN

The syntax of the LEFT JOIN is similar to that of the INNER JOIN. Let's explore the same code you used before to determine the countries with a prime minister and a president, but let's use a LEFT JOIN instead of an INNER JOIN. Further, let's remove continent to save space on the screen. The first four records in this table are the same as those from the INNER JOIN. The last six correspond to the countries that do not have a president and thus their president values are missing.

## 7. RIGHT JOIN

The RIGHT JOIN is much less common than the LEFT JOIN so we won't spend as much time on it here. The diagram will help you to understand how it works. Instead of matching entries in the id column on the left table TO the id column of the right table, a RIGHT JOIN does the reverse. Therefore, we see open circles on the ids of 5 and 6 in the right table since they are not found in the left table. The resulting table from the RIGHT JOIN shows these missing entries in the L\_val field. As you can see in SQL the right table appears after RIGHT JOIN and the left table appears after FROM.

## 8. Let's practice!

I'll see you again soon to introduce FULL JOINs after you complete the next few exercises.

**Daily XP1050**

**Exercise**

**Exercise**

**Left Join**

Now you'll explore the differences between performing an inner join and a left join using the cities and countries tables.

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.

You will then change the query to a left join. Take note of how many records are in each query here!

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))

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.

 [2](javascript:void(0))

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.

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

       region, city\_proper\_pop

FROM cities AS c1

  -- Join right table (with alias)

  \_\_\_ JOIN countries AS c2

    -- Match on country code

    ON c1.\_\_\_ = \_\_\_.code

-- Order by descending country code

ORDER BY \_\_\_ DESC;

-- Select the city name (with alias), the country code,

-- the country name (with alias), the region,

-- and the city proper population

SELECT c1.name AS \_\_\_, code, c2.name AS \_\_\_,

       region, city\_proper\_pop

-- From left table (with alias)

FROM \_\_\_ AS \_\_\_

  -- Join to right table (with alias)

  INNER JOIN \_\_\_ AS \_\_\_

    -- Match on country code

    ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_

-- Order by descending country code

ORDER BY code \_\_\_;

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

       region, city\_proper\_pop

FROM cities AS c1

  -- Join right table (with alias)

LEFT JOIN countries AS c2

    -- Match on country code

    ON c1.country\_code = c2.code

-- Order by descending country code

ORDER BY country\_code DESC;

Next, you'll try out another example comparing an inner join to its corresponding left join. Before you begin though, take note of how many records are in both the countries and languages tables below.

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 \*/.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
  + Perform an inner join and 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](javascript:void(0))

* 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.
* /\*
* Select country name AS country, the country's local name,
* the language name AS language, and
* the percent of the language spoken in the country
* \*/
* \_\_\_ c.name AS country, local\_name, l.name AS language, percent
* -- From left table (alias as c)
* FROM \_\_\_ AS \_\_\_
* -- Join to right table (alias as l)
* \_\_\_ JOIN \_\_\_ AS \_\_\_
* -- Match on fields
* ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_
* -- Order by descending country
* ORDER BY country \_\_\_;

/\*

Select country name AS country, the country's local name,

the language name AS language, and

the percent of the language spoken in the country

\*/

 SELECT c.name AS country, local\_name, l.name AS language, percent

-- From left table (alias as c)

FROM countries AS c

  -- Join to right table (alias as l)

  INNER JOIN languages AS l

    -- Match on fields

    ON c.code = l.code

-- Order by descending country

ORDER BY country DESC;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Zimbabwe | Zimbabwe | Shona | | null | |
| Zimbabwe | Zimbabwe | Venda | | null | |
| Zimbabwe | Zimbabwe | Tswana | | null | |
| Zimbabwe | Zimbabwe | Tonga | | null | |
| Zimbabwe | Zimbabwe | Sotho | | null | |
| Zimbabwe | Zimbabwe | Xhosa | | null | |
| Zimbabwe | Zimbabwe | sign | | null | |
| Zimbabwe | Zimbabwe | Shangani | | null | |
| Zimbabwe | Zimbabwe | Ndau | | null | |
| Zimbabwe | Zimbabwe | Nambya | | null | |
| Zimbabwe | Zimbabwe | Koisan | | null | |
| Zimbabwe | Zimbabwe | Kalanga | | null | |
| Zimbabwe | Zimbabwe | Chibarwe | | null | |
| Zimbabwe | Zimbabwe | Chewa | | null | |
| Zimbabwe | Zimbabwe | English | | null | |
| Zimbabwe | Zimbabwe | Ndebele | | null | |
| Zambia | Zambia | Lozi | | 5.5 | |
| Zambia | Zambia | unspecified | | 0.2 | |
| Zambia | Zambia | Other | | 9.7 | |
| Zambia | Zambia | Bisa | | 1 | |
| Zambia | Zambia | Lenje | | 1.1 | |
| Zambia | Zambia | Namwanga | | 1.2 | |
| Zambia | Zambia | Mambwe | | 1.3 | |
| Zambia | Zambia | Luvale | | 1.5 | |
| Zambia | Zambia | English | | 1.7 | |
| Zambia | Zambia | Lamba | | 1.8 | |
| Zambia | Zambia | Lala | | 1.8 | |
| Zambia | Zambia | Kaonde | | 1.8 | |
| Zambia | Zambia | Lunda | | 1.9 | |
| Zambia | Zambia | Tumbuka | | 2.5 | |
| Zambia | Zambia | Nsenga | | 2.9 | |
| Zambia | Zambia | Chewa | | 4.5 | |
| Zambia | Zambia | Tonga | | 11.4 | |
| Zambia | Zambia | Nyanja | | 14.7 | |
| Zambia | Zambia | Bembe | | 33.4 | |
| Yemen | Al-Yaman | Arabic | | null | |
| /\*  Select country name AS country, the country's local name,  the language name AS language, and  the percent of the language spoken in the country  \*/  \_\_\_ c.name AS country, local\_name, l.name AS language, percent  -- From left table (alias as c)  FROM \_\_\_ AS c    -- Join to right table (alias as l)    \_\_\_ JOIN languages AS \_\_\_      -- Match on fields      ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_  -- Order by descending country  ORDER BY \_\_\_ DESC; |  |  | |  | |
|  |  |  | |  | |
| * 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.   /\*  Select country name AS country, the country's local name,  the language name AS language, and  the percent of the language spoken in the country  \*/  SELECT c.name AS country, local\_name, l.name AS language, percent  -- From left table (alias as c)  FROM countries AS c    -- Join to right table (alias as l)  LEFT JOIN languages AS l      -- Match on fields      ON c.code = l.code  -- Order by descending country  ORDER BY country DESC;   | **code** | **name** | **continent** | **region** | **surface\_area** | **indep\_year** | **local\_name** | **gov\_form** | **capital** | **cap\_long** | **cap\_lat** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | AFG | Afghanistan | Asia | Southern and Central Asia | 652090 | 1919 | Afganistan/Afqanestan | Islamic Emirate | Kabul | 69.1761 | 34.5228 | | NLD | Netherlands | Europe | Western Europe | 41526 | 1581 | Nederland | Constitutional Monarchy | Amsterdam | 4.89095 | 52.3738 | | ALB | Albania | Europe | Southern Europe | 28748 | 1912 | Shqiperia | Republic | Tirane | 19.8172 | 41.3317 | | DZA | Algeria | Africa | Northern Africa | 2381740 | 1962 | Al-Jazair/Algerie | Republic | Algiers | 3.05097 | 36.7397 | | ASM | American Samoa | Oceania | Polynesia | 199 | null | Amerika Samoa | US Territory | Pago Pago | -170.691 | -14.2846 | | AND | Andorra | Europe | Southern Europe | 468 | 1278 | Andorra | Parliamentary Coprincipality | Andorra la Vella | 1.5218 | 42.5075 | | AGO | Angola | Africa | Central Africa | 1246700 | 1975 | Angola | Republic | Luanda | 13.242 | -8.81155 | | ATG | Antigua and Barbuda |  |  |  |  |  |  |  |  |  |   Perfect! Notice that the INNER JOIN version resulted in 909 records. The LEFT JOIN version returned 916 rows. Left join (3) You'll now revisit the use of the AVG() function introduced in our introductory SQL course. You will use it in combination with left join to determine the average gross domestic product (GDP) per capita **by region** in 2010. Instructions 1/3 **35 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * [3](javascript:void(0)) * 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.   -- Select name, region, and gdp\_percapita  SELECT \_\_\_, \_\_\_, \_\_\_  -- From countries (alias as c)  FROM \_\_\_ AS \_\_\_    -- Left join with economies (alias as e)    LEFT JOIN \_\_\_ AS \_\_\_      -- Match on code fields      ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_  -- Focus on 2010  WHERE \_\_\_ = \_\_\_;  -- Select name, region, and gdp\_percapita  SELECT name, region, gdp\_percapita  -- From countries (alias as c)  FROM countries AS c    -- Left join with economies (alias as e)    LEFT JOIN economies AS e      -- Match on code fields      ON c.code = e.code  -- Focus on 2010  WHERE year = 2010;   | **name** | **region** | **gdp\_percapita** | | --- | --- | --- | | Afghanistan | Southern and Central Asia | 539.667 | | Angola | Central Africa | 3599.27 | | Albania | Southern Europe | 4098.13 | | United Arab Emirates | Middle East | 34628.63 | | Argentina | South America | 10412.95 | | Armenia | Middle East | 3121.78 | | Antigua and Barbuda |  |  |  * 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.   -- Select fields  SELECT \_\_\_, \_\_\_ AS \_\_\_  -- From countries (alias as c)  FROM \_\_\_ AS \_\_\_    -- Left join with economies (alias as e)    LEFT JOIN \_\_\_ AS \_\_\_      -- Match on code fields      ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_  -- Focus on 2010  WHERE \_\_\_ = \_\_\_  -- Group by region  GROUP BY \_\_\_;  -- Select fields  SELECT region, AVG(gdp\_percapita) AS avg\_gdp  -- From countries (alias as c)  FROM countries AS c    -- Left join with economies (alias as e)    LEFT JOIN economies AS e      -- Match on code fields      ON c.code = e.code  -- Focus on 2010  WHERE year = 2010  -- Group by region  GROUP BY region;   | **region** | **avg\_gdp** | | --- | --- | | Southern Africa | 5051.597973632813 | | Caribbean | 11413.339454064002 | | Eastern Africa | 1757.3481616973877 | | Southern Europe | 22926.410910866478 | | Eastern Asia | 24962.8076171875 | | South America | 7420.599232991536 | | Baltic Countries | 12631.029947916666 | | North America | 47911.509765625 |   -- Select fields  SELECT region, AVG(gdp\_percapita) AS avg\_gdp  -- From countries (alias as c)  FROM countries AS c    -- Left join with economies (alias as e)    LEFT JOIN economies AS e      -- Match on code fields      ON c.code = e.code  -- Focus on 2010  WHERE year = 2010  -- Group by region  GROUP BY region  -- Order by descending avg\_gdp  ORDER BY avg\_gdp DESC;   | **region** | **avg\_gdp** | | --- | --- | | Western Europe | 58130.96149553572 | | Nordic Countries | 57073.99765625 | | North America | 47911.509765625 | | Australia and New Zealand | 44792.384765625 | | British Islands | 43588.330078125 | | Eastern Asia | 24962.8076171875 | | Southern Europe | 22926.410910866478 | | Middle East | 18204.641515395222 | | Baltic Countries | 12631.029947916666 | | Caribbean | 11413.339454064002 | | Southeast Asia |  |   Well done. Notice how gradually you're adding more and more building blocks to your SQL vocabulary. This enables you to answer questions of ever-increasing complexity!  **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.  **Instructions**  **100 XP**  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!  -- convert this code to use RIGHT JOINs instead of LEFT JOINs  /\*  SELECT cities.name AS city, urbanarea\_pop, countries.name AS country,         indep\_year, languages.name AS language, percent  FROM cities    LEFT JOIN countries      ON cities.country\_code = countries.code    LEFT JOIN languages      ON countries.code = languages.code  ORDER BY city, language;  \*/  SELECT cities.name AS city, urbanarea\_pop, countries.name AS country,         indep\_year, languages.name AS language, percent  FROM languages    RIGHT JOIN \_\_\_      ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_    RIGHT JOIN \_\_\_      ON \_\_\_.\_\_\_ = \_\_\_.\_\_\_  ORDER BY city, language;  SELECT cities.name AS city, urbanarea\_pop, countries.name AS country,         indep\_year, languages.name AS language, percent  FROM languages    RIGHT JOIN countries      ON languages.code = countries.code    RIGHT JOIN cities      ON countries.code = cities.country\_code  ORDER BY city, language;   | **city** | **urbanarea\_pop** | **country** | **indep\_year** | **language** | **percent** | | --- | --- | --- | --- | --- | --- | | Abidjan | 4765000 | Cote d'Ivoire | 1960 | French | null | | Abidjan | 4765000 | Cote d'Ivoire | 1960 | Other | null | | Abu Dhabi | 1145000 | United Arab Emirates | 1971 | Arabic | null | | Abu Dhabi | 1145000 | United Arab Emirates | 1971 | English | null | | Abu Dhabi | 1145000 | United Arab Emirates | 1971 | Hindi | null | | Abu Dhabi | 1145000 | United Arab Emirates | 1971 | Persian | null | | Abu Dhabi | 1145000 |  |  |  |  |   Correct; everything should be reversed!  **Daily XP300** FULL JOINs **50 XP** 1. FULL JOINs The last of the three types of OUTER JOINs is the FULL JOIN. In this video, you'll see the differences between a FULL JOIN and the other joins you've learned about. In particular, the instruction will focus on comparing them to INNER JOINs and LEFT JOINs and then to LEFT JOINs and RIGHT JOINs. Let's first review how the diagram changes between an INNER JOIN and a LEFT JOIN for our basic example using the left and right tables. Then we'll delve into the FULL JOIN diagram and its SQL code. 2. INNER JOIN vs LEFT JOIN Recall that an INNER JOIN keeps only the records that have matching key field values in both tables. A LEFT JOIN keeps all of the records in the left table while bringing in missing values for those key field values that don't appear in the right table. Let's next review the differences between a LEFT JOIN and a RIGHT JOIN. 3. LEFT JOIN vs RIGHT JOIN Now you can see the differences between a LEFT JOIN and a RIGHT JOIN. The id values of 2 and 3 in the left table do not match with the id values in the right table, so missing values are brought in for them in the LEFT JOIN. Likewise for the RIGHT JOIN, missing values are brought in for id values of 5 and 6. 4. FULL JOIN initial diagram A FULL JOIN combines a LEFT JOIN and a RIGHT JOIN as you can see by looking at this diagram. So it will bring in all records from both the left and the right table and keep track of the missing values accordingly. 5. FULL JOIN diagram Note the missing values here and that all six of the values of id are included in the table. You can also see from the SQL code to produce this FULL JOIN result that the general format aligns closely with the SQL syntax you've seen for both an INNER JOIN and a LEFT JOIN. You'll next explore an example from the leaders database. 6. FULL JOIN example using leaders database Let's revisit the example of looking at countries with prime ministers and/or presidents. We'll walk through the code line by line to do this using a FULL JOIN. The SELECT statement starts us off by including the country field from both of our tables of interest and also the prime\_minister and president fields. 7. FULL JOIN example using leaders database Next, the left table is specified as prime\_ministers. Note that the order matters here and if you switched the two tables you'd get slightly different output. 8. FULL JOIN example using leaders database The right table is specified as presidents with the alias of p2. prime\_ministers was aliased as p1 in the previous line. 9. FULL JOIN example using leaders database Lastly, the join is done based on the key field of country in both tables. 10. FULL JOIN example results using leaders Time for some practice! 11. Let's practice! **Daily XP350** ExerciseExerciseFull join 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. Instructions 1/3 **35 XP**   * [1](javascript:void(0))   Choose records in which region corresponds to North America or is NULL.   [2](javascript:void(0))  Repeat the same query as before, using a LEFT JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN result!   [3](javascript:void(0))  Repeat the same query again but use an INNER JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN and LEFT JOIN results!  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM \_\_\_    -- Join to currencies    FULL JOIN \_\_\_      -- Match on code      USING (\_\_\_)  -- Where region is North America or null  WHERE \_\_\_ = \_\_\_ OR \_\_\_ IS \_\_\_  -- Order by region  ORDER BY \_\_\_;  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM countries    -- Join to currencies    FULL JOIN currencies      -- Match on code      USING (code)  -- Where region is North America or null  WHERE region = 'North America' or region is NULL  -- Order by region  ORDER BY region;   | **country** | **code** | **region** | **basic\_unit** | | --- | --- | --- | --- | | Bermuda | BMU | North America | Bermudian dollar | | United States | USA | North America | United States dollar | | Canada | CAN | North America | Canadian dollar | | Greenland | GRL | North America | null | | null | TMP | null | United States dollar | | null | FLK | null | Falkland Islands pound | | null | HKG | null | Hong Kong dollar | | null | AIA | null | East Caribbean dollar | | null | NIU | null | New Zealand dollar | | null | ROM | null | Romanian leu | | null | SHN | null | Saint Helena pound | | null | SGS | null | British pound | | null | TWN | null | New Taiwan dollar | | null | WLF | null | CFP franc | | null | MSR | null | East Caribbean dollar | | null | IOT | null | United States dollar | | null | CCK | null | Australian dollar |   Repeat the same query as before, using a LEFT JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN result!  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM countries    LEFT Join to currencies    \_\_\_ \_\_\_ currencies      -- Match on code      \_\_\_ (code)  -- Where region is North America or null  WHERE region = 'North America' OR region IS NULL  -- Order by region  ORDER BY region;  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM countries    --LEFT Join to currencies    LEFT JOIN currencies      -- Match on code      USING (code)  -- Where region is North America or null  WHERE region = 'North America' OR region IS NULL  -- Order by region  ORDER BY region;   | **country** | **code** | **region** | **basic\_unit** | | --- | --- | --- | --- | | Bermuda | BMU | North America | Bermudian dollar | | Canada | CAN | North America | Canadian dollar | | United States | USA | North America | United States dollar | | Greenland | GRL | North America | null |   Repeat the same query again but use an INNER JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN and LEFT JOIN results!  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM countries    -- Join to currencies    \_\_\_ \_\_\_ currencies      -- Match on code      USING (\_\_\_)  -- Where region is North America or null  WHERE region = 'North America' OR region IS NULL  -- Order by region  ORDER BY region;  SELECT name AS country, code, region, basic\_unit  -- From countries  FROM countries    -- Join to currencies    INNER JOIN currencies      -- Match on code      USING (code)  -- Where region is North America or null  WHERE region = 'North America' OR region IS NULL  -- Order by region  ORDER BY region;   | **country** | **code** | **region** | **basic\_unit** | | --- | --- | --- | --- | | Bermuda | BMU | North America | Bermudian dollar | | Canada | CAN | North America | Canadian dollar | | United States | USA | North America | United States dollar |   Have you kept an eye out on the different numbers of records these queries returned**? The FULL JOIN query returned 18 rows, the OUTER JOIN returned 4 rows, and the INNER JOIN only returned 3 rows. Do these results make sense to you?**  **Daily XP450** ExerciseExerciseFull join (2) 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. Instructions 1/3 **35 XP**   * [1](javascript:void(0))   + 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](javascript:void(0))   * Repeat the same query as before, using a LEFT JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN result!    [3](javascript:void(0))   * Repeat once more, but use an INNER JOIN instead of a LEFT JOIN. Note what has changed compared to the FULL JOIN and LEFT JOI   SELECT countries.name, code, languages.name AS language  -- From languages  FROM \_\_\_    -- Join to countries    \_\_\_ JOIN \_\_\_      -- Match on code      USING (\_\_\_)  -- Where countries.name starts with V or is null  WHERE \_\_\_.\_\_\_ LIKE \_\_\_ OR \_\_\_.\_\_\_ IS \_\_\_  -- Order by ascending countries.name  ORDER BY \_\_\_.\_\_\_;  SELECT countries.name, code, languages.name AS language  -- From languages  FROM languages    -- Join to countries    FULL JOIN countries      -- Match on code      USING (code)  -- Where countries.name starts with V or is null  WHERE countries.name LIKE 'V%' OR countries.name IS NULL  -- Order by ascending countries.name  ORDER BY countries.name;   | **name** | **code** | **language** | | --- | --- | --- | | Vanuatu | VUT | Tribal Languages | | Vanuatu | VUT | English | | Vanuatu | VUT | French | | Vanuatu | VUT | Other | | Vanuatu | VUT | Bislama | | Venezuela | VEN | Spanish | | Venezuela | VEN | indigenous | | Vietnam | VNM | Vietnamese | | Vietnam | VNM | English | | Vietnam | VNM | Other | | Virgin Islands, British | VGB | null | | Virgin Islands, U.S. | VIR | null |   SELECT countries.name, code, languages.name AS language  -- From languages  FROM languages    -- Join to countries    LEFT JOIN countries      -- Match on code      USING (code)  -- Where countries.name starts with V or is null  WHERE countries.name LIKE 'V%' OR countries.name IS NULL  -- Order by ascending countries.name  ORDER BY countries.name;   | **name** | **code** | **language** | | --- | --- | --- | | Vanuatu | VUT | English | | Vanuatu | VUT | Other | | Vanuatu | VUT | French | | Vanuatu | VUT | Tribal Languages | | Vanuatu | VUT | Bislama | | Venezuela | VEN | indigenous | | Venezuela | VEN | Spanish |   SELECT countries.name, code, languages.name AS language  -- From languages  FROM languages    -- Join to countries    \_\_\_ \_\_\_ countries      -- Match using code      USING (code)  -- Where countries.name starts with V or is null  WHERE countries.name LIKE 'V%' OR countries.name IS NULL  -- Order by ascending countries.name  \_\_\_ \_\_\_ countries.name; |  |  | |  | |
| 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.  SELECT countries.name, code, languages.name AS language  -- From languages  FROM languages    -- Join to countries    INNER JOIN countries      -- Match using code      USING (code)  -- Where countries.name starts with V or is null  WHERE countries.name LIKE 'V%' OR countries.name IS NULL  -- Order by ascending countries.name  ORDER BY countries.name;   | **name** | **code** | **language** | | --- | --- | --- | | Vanuatu | VUT | Tribal Languages | | Vanuatu | VUT | Bislama | | Vanuatu | VUT | English | | Vanuatu | VUT | French | | Vanuatu | VUT | Other | | Venezuela | VEN | Spanish | | Venezuela | VEN | indigenous | | Vietnam | VNM | Vietnamese |   **Daily XP550**  **Exercise**  **Exercise**  **Full join (2)**  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.  **Instructions 3/3**  **30 XP**   * + 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.   + Repeat the same query as before, using a LEFT JOIN instead of a FULL JOIN. Note what has changed compared to the FULL JOIN result!   + 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.   +100 XP  Well done. Again, make sure to compare the number of records the different types of joins return and try to verify whether the results make sense.  **Daily XP550**  **Exercise**  **Exercise**  **Full join (3)**  You'll now explore using two consecutive full joins on the three tables you worked with in the previous two exercises.  **Instructions**  **100 XP**   * 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.   -- Select fields (with aliases)  SELECT \_\_\_.\_\_\_ AS \_\_\_, \_\_\_, \_\_\_.\_\_\_ AS \_\_\_,         \_\_\_, \_\_\_  -- From countries (alias as c1)  FROM \_\_\_ AS \_\_\_    -- Join with languages (alias as l)    FULL JOIN \_\_\_ AS \_\_\_      -- Match on code      USING (\_\_\_)    -- Join with currencies (alias as c2)    FULL JOIN \_\_\_ AS \_\_\_      -- Match on code      USING (\_\_\_)  -- Where region like Melanesia and Micronesia  WHERE \_\_\_ LIKE \_\_\_;  -- Select fields (with aliases)  SELECT c1.name AS country, region, l.name AS language,         basic\_unit, frac\_unit  -- From countries (alias as c1)  FROM countries AS c1    -- Join with languages (alias as l)    FULL JOIN languages AS l      -- Match on code      USING (code)    -- Join with currencies (alias as c2)    FULL JOIN currencies AS c2      -- Match on code      USING (code)  -- Where region like Melanesia and Micronesia  WHERE region LIKE 'M%esia';   | **country** | **region** | **language** | **basic\_unit** | **frac\_unit** | | --- | --- | --- | --- | --- | | Kiribati | Micronesia | English | Australian dollar | Cent | | Kiribati | Micronesia | Kiribati | Australian dollar | Cent | | Marshall Islands | Micronesia | Other | United States dollar | Cent | | Marshall Islands | Micronesia | Marshallese | United States dollar | Cent | | Nauru | Micronesia | Other | Australian dollar | Cent | | Nauru | Micronesia | English | Australian dollar | Cent | | Nauru | Micronesia | Nauruan | Australian dollar | Cent | | New Caledonia | Melanesia | Other |  |  |   **Review outer joins**  **A(n) FULL join is a join combining the results of a LEFT join and a RIGHT join.**  **Answer the question**  **50XP**  **Possible Answers**      left, full, right  press1      right, full, left  press2      inner, left, right  press3      **None of the above are true**  press4 |  |  | |  | |  | | |  |  |
| 1. CROSSing the Rubicon Now that you've worked with INNER JOINs and OUTER JOINs it's time to check out the CROSS JOIN. CROSS JOINs create all possible combinations of two tables. Let's explore the diagram for a CROSS JOIN next. 2. CROSS JOIN diagram In this diagram we have two tables named table1 and table2. Each table only has one field, both with the name of id. The result of the CROSS JOIN is all nine combinations of the id values of 1, 2, and 3 in table1 with the id values of A, B, and C for table2. Next you'll explore an example from the leaders database and look over the SQL syntax for a CROSS JOIN. 3. Pairing prime ministers with presidents Suppose that all prime ministers in North America and Oceania in the prime\_ministers table are scheduled for individual meetings with all presidents in the presidents table. You can look at all of these combinations by using a CROSS JOIN. The syntax here remains similar to what you've seen earlier in the course. We use a WHERE clause to focus on only prime ministers in North America and Oceania in the prime\_ministers table. The results of the query give us the pairings for the two prime ministers in North America and Oceania from the prime\_ministers table with the seven presidents in the presidents table. 4. Let's practice! You'll now hop into an exercise focusing on a couple of cities in a tribute to the author Charles Dickens. This chapter closes with a challenge to test your comprehension of the content covered here. Good luck!  **Daily XP200** ExerciseExerciseA table of two cities This exercise looks to explore languages potentially and most frequently spoken in the cities of Hyderabad, India and Hyderabad, Pakistan. Instructions 1/2 **50 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * Create a CROSS JOIN with cities AS c on the left and languages AS l on the right. * Make use of LIKE and Hyder% to choose Hyderabad in both countries. * Select only the city name AS city and language name AS language.   -- Select fields  SELECT \_\_\_.\_\_\_ AS \_\_\_, \_\_\_.\_\_\_ AS \_\_\_  -- From cities (alias as c)  FROM \_\_\_ AS c    -- Join to languages (alias as l)    CROSS JOIN \_\_\_ AS \_\_\_  -- Where c.name like Hyderabad  WHERE \_\_\_.\_\_\_ LIKE \_\_\_;  -- Select fields  SELECT c.name AS city, l.name AS language  -- From cities (alias as c)  FROM cities AS c    -- Join to languages (alias as l)    CROSS JOIN languages AS l  -- Where c.name like Hyderabad  WHERE c.name LIKE 'Hyder%';   | **city** | **language** | | --- | --- | | Hyderabad (India) | Dari | | Hyderabad | Dari | | Hyderabad (India) | Pashto | | Hyderabad | Pashto | | Hyderabad (India) | Turkic | | Hyderabad | Turkic | | Hyderabad (India) | Other | | Hyderabad | Other | | Hyderabad (India) | Albanian | | Hyderabad | Albanian |  * 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.   -- Select fields  \_\_\_  -- From cities (alias as c)  \_\_\_    -- Join to languages (alias as l)    \_\_\_ JOIN \_\_\_ AS \_\_\_      -- Match on country code      \_\_\_  -- Where c.name like Hyderabad  \_\_\_;  -- Select fields  SELECT c.name AS city, l.name AS language  -- From cities (alias as c)  FROM cities AS c    -- Join to languages (alias as l)    INNER JOIN languages AS l      -- Match on country code     ON c.country\_code = l.code  -- Where c.name like Hyderabad  WHERE  c.name LIKE 'Hyder%';   | **city** | **language** | | --- | --- | | Hyderabad (India) | Hindi | | Hyderabad (India) | Bengali | | Hyderabad (India) | Telugu | | Hyderabad (India) | Marathi | | Hyderabad (India) | Tamil | | Hyderabad (India) | Urdu | | Hyderabad (India) | Gujarati | | Hyderabad (India) | Kannada | | Hyderabad (India) | Malayalam | | Hyderabad (India) | Oriya | | Hyderabad (India) | Punjabi | | Hyderabad (India) | Assamese | | Hyderabad (India) | Maithili | | Hyderabad (India) | Other | | Hyderabad | Punjabi | | Hyderabad | Sindhi | | Hyderabad |  |   Good one! Can you see the difference between a CROSS JOIN and a INNER JOIN?  **Daily XP300**  **Exercise**  **Exercise**  **Outer challenge**  Now that you're fully equipped to use OUTER JOINs, try a challenge problem to test your knowledge!  In terms of life expectancy for 2010, determine the names of the lowest five countries and their regions.  **Instructions**  **100 XP**   * Select country name AS country, region, and life expectancy AS life\_exp. * Make sure to use LEFT JOIN, WHERE, ORDER BY, and LIMIT.   -- Select fields  \_\_\_  -- From countries (alias as c)  \_\_\_    -- Join to populations (alias as p)    \_\_\_      -- Match on country code      \_\_\_  -- Focus on 2010  \_\_\_  -- Order by life\_exp  \_\_\_  -- Limit to 5 records  \_\_\_  -- Select fields  SELECT  c.name AS country, region, life\_expectancy AS life\_exp  -- From countries (alias as c)  FROM countries AS c    -- Join to populations (alias as p)    LEFT JOIN populations AS p      -- Match on country code      ON c.code = p.country\_code  -- Focus on 2010  WHERE year = 2010  -- Order by life\_exp  ORDER BY life\_expectancy  -- Limit to 5 records  LIMIT 5;   | **country** | **region** | **life\_exp** | | --- | --- | --- | | Lesotho | Southern Africa | 47.483414 | | Central African Republic | Central Africa | 47.625317 | | Sierra Leone | Western Africa | 48.22895 | | Swaziland | Southern Africa | 48.345757 | | Zimbabwe |  |  |   his was the last exercise of this chapter on outer joins and cross joins. In the next chapter, you'll learn about set theory clauses!  **Daily XP400** State of the UNION **50 XP** 1. State of the UNION Welcome to Chapter 3! You've made it through some challenging material so far. Keep up the great work! Next are set theory clauses. In this video, we'll focus on the operations of UNION and UNION ALL. In addition to joining diagrams, in this chapter, you'll also see how Venn diagrams can be used to represent set operations. Let's begin with these Venn diagrams now. 2. Set Theory Venn Diagrams You can think of each circle as representing a table of data. The shading represents what is included in the result of the set operation from each table. Beginning in the top left, UNION includes every record in both tables but DOES NOT double count those that are in both tables. UNION ALL includes every record in both tables and DOES replicate those that are in both tables. This is why the center is shaded black. The two diagrams on the bottom represent only subsets of data being selected. INTERSECT results in only those records found in both of the two tables. EXCEPT results in only those records in one table BUT NOT the other. Let's investigate what UNION looks like as a joining diagram. 3. UNION diagram In this diagram, you have two tables with names left\_one and right\_one. The "one" here corresponds to each table only having one field. If you run a UNION on these two fields you get each record appearing in either table, but notice that the id values of 1 and 4 in right\_one are not included again in the UNION since they were already found in the left\_one table. 4. UNION ALL diagram By contrast (with the same two tables left\_one and right\_one), UNION ALL includes all duplicates in its result. So left\_one and right\_one both having four records yields eight records for the result of the UNION ALL. If it were the case that right\_one had these same four values and also one more value of 1 for id, you'd see three entries for 1 in the resulting UNION ALL. Let's check out the SQL syntax using the leaders database for both UNION and UNION ALL, but first you'll see one more table in the leaders database. 5. monarchs table Check out the monarchs table in the leaders database that we will use in examples here. The table lists the country, continent, and the name of the monarch for that country. Do some of these names look familiar based on the other tables you've seen? They should! We'll come back to this. 6. All prime ministers and monarchs You can use UNION on the prime\_ministers and monarchs table to show all of the different prime ministers and monarchs in these two tables. The country field is also included here for reference. Note that the prime\_minister field has been aliased as leader. In fact, the resulting field from the UNION will have the name of leader. That's an important property of the set theory clauses you will see in this chapter. The fields included in the operation must be of the same data type since they come back as just a single field. You can't stack a number on top of a character field in other words. 7. Resulting table from UNION Our resulting table from the UNION gives all the leaders and their corresponding country. Does something stand out to you here? (PAUSE) 8. UNION ALL with leaders The countries of Brunei and Oman were listed only once in the UNION table. These countries have monarchs that also act as prime ministers. This can be seen in the UNION ALL results. You've seen 9. Let's practice! that UNION and UNION ALL clauses do not do the lookup step that JOINs do. They simply stack records on top of each other from one table to the next.  **Daily XP450**  **Exercise**  **Exercise**  **Union**  You have two new tables, economies2010 and economies2015, available to you. The economies table is also included for reference.  **Instructions**  **100 XP**   * Combine the two new tables into one table containing all of the fields in economies2010. * Sort this resulting single table by country code and then by year, both in ascending order.   -- Select fields from 2010 table  \_\_\_ \_\_\_    -- From 2010 table    \_\_\_ \_\_\_    -- Set theory clause    \_\_\_  -- Select fields from 2015 table  \_\_\_ \_\_\_    -- From 2015 table    \_\_\_ \_\_\_  -- Order by code and year  \_\_\_ \_\_\_ \_\_\_, \_\_\_; |  |  | |  | |
|  |  |  |  | |  | |  |

-- Select fields from 2010 table

Select code,year, income\_group, gross\_savings

  -- From 2010 table

  FROM economies2010

  -- Set theory clause

  UNION

-- Select fields from 2015 table

SELECT code, year, income\_group, gross\_savings

  -- From 2015 table

  FROM economies2015

-- Order by code and year

ORDER BY code, year;

| **code** | **year** | **income\_group** | **gross\_savings** |
| --- | --- | --- | --- |
| AFG | 2010 | Low income | 37.133 |
| AFG | 2015 | Low income | 21.466 |
| AGO | 2010 | Upper middle income | 23.534 |
| AGO | 2015 | Upper middle income | -0.425 |
| ALB | 2010 | Upper middle income | 20.011 |
| ALB | 2015 | Upper middle income | 13.84 |
| ARE | 2010 | High income | 27.073 |
| ARE | 2015 | High income | 34.106 |
| ARG | 2010 | Upper middle income | 17.361 |
| ARG | 2015 | Upper middle income | 14.111 |
| ARM | 2010 | Lower middle income | 15.797 |
| ARM | 2015 | Lower middle income | 18.306 |
| ATG | 2010 | High income |  |

**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.

**Instructions**

**100 XP**

* 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.
* -- Select field
* \_\_\_
* -- From cities
* \_\_\_
* -- Set theory clause
* \_\_\_
* -- Select field
* \_\_\_
* -- From currencies
* \_\_\_
* -- Order by country\_code
* \_\_\_;

-- Select field

SELECT c1.country\_code

  -- From cities

  FROM cities AS c1

  -- Set theory clause

  UNION

-- Select field

SELECT c2.code

  -- From currencies

  FROM currencies AS c2

-- Order by country\_code

ORDER BY country\_code;

| **country\_code** |
| --- |
| ABW |
| AFG |
| AGO |
| AIA |
| ALB |
| AND |
| ARE |
| ARG |
| ARM |

**Daily XP650**

**Exercise**

**Exercise**

**Union all**

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

To include duplicates, you can use UNION ALL.

**Instructions**

**100 XP**

* 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.

-- Select fields

SELECT \_\_\_, \_\_\_

  -- From economies

  FROM \_\_\_

  -- Set theory clause

  \_\_\_ \_\_\_

-- Select fields

SELECT \_\_\_, \_\_\_

  -- From populations

  FROM \_\_\_

-- Order by code, year

ORDER BY code, year;

-- Select fields

SELECT code, year

  -- From economies

  FROM economies

  -- Set theory clause

  UNION ALL

-- Select fields

SELECT country\_code, year

  -- From populations

  FROM populations

-- Order by code, year

ORDER BY code, year;

| **code** | **year** |
| --- | --- |
| ABW | 2010 |
| ABW | 2015 |
| AFG | 2010 |
| AFG | 2010 |
| AFG | 2015 |
| AFG | 2015 |
| AGO | 2010 |
| AGO |  |

Can you spot some duplicates in the query result?

## 1. INTERSECTional data science

You saw with UNION and UNION ALL via examples that they do not do quite the same thing as what a join does. They only bind fields on top of one another in the two tables. The set theory clause INTERSECT works in a similar fashion to UNION and UNION ALL, but remember from the Venn diagram that INTERSECT only includes those records in common to both tables and fields selected. Let's investigate the diagram for INTERSECT and the corresponding SQL code to achieve it.

## 2. INTERSECT diagram and SQL code

The result of the INTERSECT on left\_one and right\_one is only the records in common to both left\_one and right\_one: 1 and 4. Let's next see how you could use INTERSECT to determine all countries having both a prime minister and a president.

## 3. Prime minister and president countries

The code for each of these set operations has a similar layout. You first select which fields you'd like to include in your first table, and then you specify the name of the first table. Next you specify the set operation to perform. Lastly, you denote which fields in the second table you'd like to include and then the name of the second table. The result of the query is the four countries with both a prime minister and a president in the leaders database.

## 4. INTERSECT on two fields

Next, let's think about what would happen if we tried to select two columns instead of one from our previous example. The code shown does just that. What will be the result of this query? Will this also give you the names of the countries that have both a prime minister and a president? Hmmm [PAUSE] The actual result is an empty table. Why is that? When INTERSECT looks at two columns it includes both columns in the search. So it didn't find any countries with prime ministers AND presidents having the same name. INTERSECT looks for RECORDS in common, not individual key fields like what a join does to match. This is an important distinction.

## 5. Let's practice!

Let's get some practice!

**Daily XP800**

**Exercise**

**Exercise**

**Intersect**

UNION ALL will extract all records from two tables, while INTERSECT will only return records that both tables have in common. In this exercise, you will create a similar query as before, however, this time you will look at the records in common for country code and year for the economies and populations tables.

Note the number of records from the result of this query compared to the similar UNION ALL query result of 814 records.

**Instructions**

**100 XP**

* Use INTERSECT to determine 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.

-- Select fields

\_\_\_

  -- From economies

  \_\_\_

  -- Set theory clause

  \_\_\_

-- Select fields

\_\_\_

  -- From populations

  \_\_\_

-- Order by code and year

\_\_\_;

-- Select fields

SELECT code, year

  -- From economies

  FROM economies

  -- Set theory clause

  INTERSECT

-- Select fields

SELECT country\_code, year

  -- From populations

  FROM populations

-- Order by code and year

ORDER BY code, year;

| **code** | **year** |
| --- | --- |
| AFG | 2010 |
| AFG | 2015 |
| AGO | 2010 |
| AGO | 2015 |
| ALB | 2010 |
| ALB | 2015 |
| ARE | 2010 |
| ARE | 2015 |
| ARG | 2010 |
| ARG | 2015 |

**Daily XP900**

**Exercise**

**Exercise**

**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?*

**Instructions**

**100 XP**

Use INTERSECT to answer this question with countries and cities!

-- Select fields

\_\_\_

  -- From countries

  \_\_\_

  -- Set theory clause

  \_\_\_

-- Select fields

\_\_\_

  -- From cities

  \_\_\_;

-- Select fields

SELECT name

  -- From countries

  FROM countries

  -- Set theory clause

  INTERSECT

-- Select fields

SELECT name

  -- From cities

  FROM cities;

| **name** |
| --- |
| Singapore |

Nice one! It looks as though Singapore is the only country that has a city with the same name!

**Review union and intersect**

Which of the following combinations of terms and definitions is correct?

**Answer the question**

**50XP**

**Possible Answers**

* 

UNION: returns all records (potentially duplicates) in both tables

press1

* 

UNION ALL: returns only unique records

press2

* 

**INTERSECT: returns only records appearing in both tables**

press3

* 

None of the above are matched correctly

press4

## 1. EXCEPTional

Way to go! You've made it to the last of the four set theory clauses in this course. EXCEPT allows you to include only the records that are in one table, but not the other. Let's mix things up and look into the SQL code and result first and then dive into the diagram.

## 2. Monarchs that aren't prime ministers

You saw earlier that there are some monarchs that also act as the prime minister for their country. One way to determine those monarchs in the monarchs table that do not also hold the title of prime minister is to use the EXCEPT clause. [CLICK] This SQL query selects the monarch field from monarchs and then looks for common entries with the prime\_ministers field, while also keeping track of the country for each leader. [CLICK] You can see in the resulting query that only the two European monarchs are not also prime ministers in the leaders database.

## 3. EXCEPT diagram

This diagram gives the structure of EXCEPT clauses. Only the records that appear in the left table BUT DO NOT appear in the right table are included.

## 4. Let's practice!

After a couple exercises on using EXCEPT clauses, you'll check out the last two types of joins for the course: semi-joins and anti-joins. I'll see you in the next video for them!

**Daily XP1100**

**Exercise**

**Exercise**

**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.

**Instructions**

**100 XP**

* Order the resulting field in ascending order.
* Can you spot the city/cities that are actually capital cities which this query misses?

-- Select field

SELECT \_\_\_

  -- From cities

  FROM \_\_\_

  -- Set theory clause

  \_\_\_

-- Select field

SELECT \_\_\_

  -- From countries

  FROM \_\_\_

-- Order by result

ORDER BY \_\_\_;

- Select field

SELECT name

  -- From cities

  FROM cities

  -- Set theory clause

  EXCEPT

-- Select field

SELECT capital

  -- From countries

  FROM countries

-- Order by result

ORDER BY name;

| **name** |
| --- |
| Abidjan |
| Ahmedabad |
| Alexandria |
| Almaty |
| Auckland |
| Bandung |
| Barcelona |
| Barranquilla |
| Basra |
| Belo Horizonte |
| Bengaluru |
| Bhopal |
| Birmingham |
| Brisbane |
| Bucharest |
| Busan |
| Calgary |
| Cali |
| Caloocan |
| Campinas |

**Daily XP1200**

**Exercise**

**Exercise**

**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.

**Instructions**

**100 XP**

* 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!

-- Select field

\_\_\_

  -- From countries

  \_\_\_

  -- Set theory clause

  \_\_\_

-- Select field

\_\_\_

  -- From cities

  \_\_\_

-- Order by ascending capital

\_\_\_;

Well done. Is this query surprising, as the instructions suggested?

-- Select field

SELECT capital

  -- From countries

  FROM countries

  -- Set theory clause

  EXCEPT

-- Select field

SELECT name

  -- From cities

  FROM cities

-- Order by ascending capital

ORDER BY capital;

| **capital** |
| --- |
| Agana |
| Amman |
| Amsterdam |
| Andorra la Vella |
| Antananarivo |
| Apia |
| Ashgabat |
| Asmara |
| Astana |
| Asuncion |
| Athens |
| Bamako |
| Bandar Seri Begawan |
| Bangui |
| Banjul |
| Basseterre |
| Beirut |
| Belmopan |
| Bern |
| Bishkek |
| Bissau |
| Bratislava |
| Bridgetown |
| Brussels |
| Bujumbura |
| Canberra |
| Castries |
| Charlotte Amalie |
| Chisinau |

# Semi-joins and Anti-joins

**50 XP**

## 1. Semi-joins and Anti-joins

You'll now close this chapter by returning to joins. The six joins you've worked with so far are all additive joins in that they add columns to the original "left" table. Can you name all six? (1. INNER JOIN, 2. self-join, 3. LEFT JOIN, 4. RIGHT JOIN, 5. FULL JOIN, and 6. CROSS JOIN).

## 2. Building up to a semi-join

The last two joins we will cover use a right table to determine which records to keep in the left table. In other words, you use these last two joins (semi-join and anti-join) in a way similar to a WHERE clause dependent on the values of a second table. Let's try out some examples of semi-joins and anti-joins and then return to the diagrams for each. Suppose that you are interested in determining the presidents of countries that gained independence before 1800. Let's first determine which countries this corresponds to in the states table. Recall from your knowledge of SQL before you knew anything about JOINs how this could be done. To get only the countries meeting this condition you can use the WHERE clause. We'll next set up the other part of the query to get the

## 3. Another step towards the semi-join

presidents we want. What code is needed to retrieve the president, country, and continent columns from the presidents table in that order? [PAUSE] Now we need to use this result with the one in the previous slide to further filter the country field in the presidents table to give us the correct result. Let's see how this might be done next.

## 4. Finish the semi-join (an intro to subqueries)

In the first query of this example, we determined that Portugal and Spain were both independent before 1800. In the second query, we determined how to display the table in a nice form to answer our question. In order to combine the two tables together we will again use a WHERE clause and then use the first query as the condition to check in the WHERE clause. Check it out! This is your first example of a subquery: a query that sits inside of another query. You'll explore these more in Chapter 4. What does this give as a result? Is it the presidents of Spain and of Portugal? Since Spain does not have a president, it is not included here and only the Portuguese president is listed. The semi-join chooses records in the first table where a condition IS met in a second table. An anti-join chooses records in the first table where a condition IS NOT met in the second table. How might you determine countries in the Americas founded after 1800?

## 5. An anti-join

Using the code from the previous example, you only need to add a few pieces of code. So what goes in the blanks? [PAUSE] Fill in the WHERE clause by choosing only those continents ending in America and then fill in the other space with a NOT to exclude those countries in the subquery. The presidents of

## 6. The result of the anti-join

countries in the Americas founded after 1800 are given in the table.

## 7. Semi-join and anti-join diagrams

The semi-join matches records by key field in the right table with those in the left. It then picks out only the rows in the left table that match that condition. The anti-join picks out those columns in the left table that do not match the condition on the right table. Semi-joins and anti-joins don't have the same built-in SQL syntax that INNER JOIN and LEFT JOIN have. They are useful tools in filtering one table's records on the records of another table.

## 8. Let's practice!

This chapter's challenge exercise will ask you to combine set theory clauses with semi-joins. Before you get to that, you'll try out some exercises on semi-joins and anti-joins. See you again in Chapter 4!

**Daily XP1350**

##### Exercise

##### Exercise

# Semi-join

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

##### Instructions 1/3

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

Begin by selecting all country codes in the Middle East as a single field result using SELECT, FROM, and WHERE.

-- Select code

\_\_\_

  -- From countries

  \_\_\_

-- Where region is Middle East

\_\_\_;

-- Select code

SELECT code

  -- From countries

  FROM countries

-- Where region is Middle East

WHERE region = 'Middle East';

| **code** |
| --- |
| ARE |
| ARM |
| AZE |
| BHR |
| GEO |
| IRQ |
| ISR |
| YEM |

* 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.
* -- Query from step 1:
* /\*
* SELECT code
* FROM countries
* WHERE region = 'Middle East';
* \*/
* -- Select field
* \_\_\_
* -- From languages
* \_\_\_
* -- Order by name
* \_\_\_;
* -- Query from step 1:
* /\*
* SELECT code
* FROM countries
* WHERE region = 'Middle East';
* \*/
* -- Select field
* SELECT DISTINCT name
* -- From languages
* FROM languages
* -- Order by name
* ORDER BY name;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **name** | | --- | | Afar | | Afrikaans | | Akyem | | Albanian | | Alsatian | | Amerindian | | Amharic | | Angolar |   -- Query from step 2  SELECT DISTINCT name    FROM languages  -- Where in statement  WHERE code IN    -- Query from step 1    -- Subquery    (SELECT code     FROM countries     WHERE region = 'Middle East')  -- Order by name  ORDER BY name;   | **name** | | --- | | Arabic | | Aramaic | | Armenian | | Azerbaijani | | Azeri | | Baluchi | | Bulgarian | | Circassian |   **Daily XP1450**  **Exercise**  **Exercise**  **Relating semi-join to a tweaked inner join**  Let's revisit the code from the previous exercise, which retrieves languages spoken in the Middle East.  SELECT DISTINCT name  FROM languages  WHERE code IN  (SELECT code  FROM countries  WHERE region = 'Middle East')  ORDER BY name;  Sometimes problems solved with semi-joins can also be solved using an inner join.  SELECT **DISTINCT**languages.name AS language  FROM languages  INNER JOIN countries  ON languages.code = countries.code  WHERE region = 'Middle East'  ORDER BY language;  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?  **Instructions**  **50 XP**  **Possible Answers**      HAVING instead of WHERE      **DISTINCT**      UNIQUE  Correct! There's no use on retrieving 'Arabic' multiple times; you only care about DISTINCT languages here.  **Daily XP1500** ExerciseExerciseDiagnosing 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! Instructions 1/3 **35 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * [3](javascript:void(0))   Begin by determining the number of countries in countries that are listed in Oceania using SELECT, FROM, and WHERE.  -- Select statement  \_\_\_    -- From countries    \_\_\_  -- Where continent is Oceania  \_\_\_;  -- Select statement  SELECT COUNT(name)    -- From countries    FROM countries  -- Where continent is Oceania  WHERE continent = 'Oceania';   | **count** | | --- | | 19 |  * 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 the query result and make note of how many *different* countries are listed here.  -- Select fields (with aliases)  \_\_\_    -- From countries (alias as c1)    \_\_\_      -- Join with currencies (alias as c2)      \_\_\_      -- Match on code      \_\_\_  -- Where continent is Oceania  \_\_\_;  -- Select fields (with aliases)  SELECT c1.code, c1.name, basic\_unit AS currency    -- From countries (alias as c1)    FROM countries AS c1      -- Join with currencies (alias as c2)      INNER JOIN currencies AS c2      -- Match on code      ON c1.code = c2.code  -- Where continent is Oceania  WHERE continent = 'Oceania';   | **code** | **name** | **currency** | | --- | --- | --- | | AUS | Australia | Australian dollar | | PYF | French Polynesia | CFP franc | | KIR | Kiribati | Australian dollar | | MHL | Marshall Islands | United States dollar | | NRU | Nauru | Australian dollar | | NCL | New Caledonia | CFP franc | | NZL | New Zealand | New Zealand dollar | | PLW | Palau | United States dollar | | PNG | Papua New Guinea | Papua New Guinean kina | | WSM | Samoa | Samoan tala | | SLB | Solomon Islands | Solomon Islands dollar | | TON | Tonga | Tongan paʻanga | | TUV | Tuvalu | Australian dollar | | TUV | Tuvalu |  |   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.   -- Select fields  \_\_\_    -- From Countries    \_\_\_    -- Where continent is Oceania    \_\_\_      -- And code not in      \_\_\_      -- Subquery      (\_\_\_       \_\_\_);  -- Select fields  SELECT code, name    -- From Countries    FROM countries    -- Where continent is Oceania    WHERE continent = 'Oceania'      -- And code not in      AND code NOT IN      -- Subquery      (SELECT code       FROM currencies);   |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | | **code** | **name** | | --- | --- | | ASM | American Samoa | | FJI | Fiji Islands | | GUM | Guam | | FSM | Micronesia, Federated States of | | MNP | Northern Mariana Isla | |  | | Nice! Can you tell which countries were not included now?  **Daily XP1600**  **Exercise**  **Exercise**  **Set theory challenge**  Congratulations! You've now made your way to the challenge problem for this third chapter. 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!  **Instructions**  **100 XP**   * 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.   -- Select the city name  \_\_\_ name    -- Alias the table where city name resides    \_\_\_ \_\_\_ AS c1    -- Choose only records matching the result of multiple set theory clauses    WHERE \_\_\_ IN  (      -- Select appropriate field from economies AS e      SELECT \_\_\_.\_\_\_      FROM \_\_\_ AS \_\_\_      -- Get all additional (unique) values of the field from currencies AS c2      \_\_\_      SELECT \_\_\_.\_\_\_      FROM \_\_\_ AS \_\_\_      -- Exclude those appearing in populations AS p      \_\_\_      SELECT \_\_\_.\_\_\_      FROM \_\_\_ AS \_\_\_  );  -- Select the city name  SELECT c1.name    -- Alias the table where city name resides    FROM  cities AS c1    -- Choose only records matching the result of multiple set theory clauses    WHERE c1.country\_code IN  (      -- Select appropriate field from economies AS e      SELECT e.code      FROM economies AS e      -- Get all additional (unique) values of the field from currencies AS c2      UNION      SELECT c2.code      FROM currencies AS c2      -- Exclude those appearing in populations AS p      EXCEPT      SELECT p.country\_code      FROM populations AS p  );   | **name** | | --- | | Bucharest | | Kaohsiung | | New Taipei City | | Taichung | | Tainan | | Taipei |   **Daily XP1700** ExerciseExerciseSet theory challenge Congratulations! You've now made your way to the challenge problem for this third chapter. 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! Instructions **100 XP**   * 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.   Incorrect submission Check the highlighted code. The checker expected to find country\_code in there.  Did you find this feedback helpful?  +100 XP  Success! Head over to the final chapter of this course to feel the power of subqueries at your fingertips!  **Daily XP1700** Subqueries inside WHERE and SELECT clauses **50 XP** 1. Subqueries inside WHERE and SELECT clauses You've made it all the way to the last chapter of this course. Congratulations and keep up the excellent work. This last chapter is focused on embedding queries inside of queries. These are called nested queries and also known as subqueries as you saw in Chapter 3. The most common type of subquery is one inside of a WHERE statement. Let's check out another one of these now with a little bit of setting up to do first, of course. 2. Subquery inside WHERE clause set-up You've seen many examples of using a subquery inside a WHERE clause already with the semi-join and anti-join examples and exercises you just completed. With the WHERE clause being the most common place for a subquery to be found, it's important that you see just one more example of doing so. With this being the final chapter, it's time to unveil the remaining fields in the states table. Note that the continent field is not shown to display all the fields here. The fert\_rate field gives an estimate for the average number of babies born per woman in each country. The women\_parli\_perc field gives the percentage of women in the elected federal parliament for each country. Across these 13 countries, how would you determine the average fertility rate? 3. Average fert\_rate We will use the average fertility rate as part of a subquery. Recall how this is done. The average babies born to women across these countries is 2-point-28 children. 4. Asian countries below average `fert\_rate` Let's use the previous slide's query as a subquery to determine Asian countries that fall under this average. You'll see the code in a couple steps. First we select the country name and the fertility rate for Asian countries. 5. Asian countries below average `fert\_rate` Next, we want to choose records where fert\_rate is smaller than What comes next? 6. Asian countries below average `fert\_rate` The subquery is to get the average fertility rate! And now we can check out our result to make sure it makes sense. 7. Asian countries below average `fert\_rate` It appears so. These are the two Asian countries we were looking for with fertility rates below 2-point-28 babies per woman. 8. Subqueries inside SELECT clauses - setup The second most common type of a subquery is inside of a SELECT clause. The task here is to count the number of countries listed in the states table for each continent in the prime\_ministers table. Let's again take the stepwise approach to setting up the problem. What does this code do? [PAUSE] It gives each of the five continents in the prime\_ministers table. Let's keep building our answer in the next slide. 9. Subquery inside SELECT clause - complete Next is determining the counts of the number of countries in states for each of the continents in the last slide. Combining a COUNT clause with a WHERE statement matching the continent fields in the two tables gets us there. Let's check out the code and then discuss a bit further. The subquery involving states also can reference the prime\_ministers table in the main query. Any time you do a subquery inside a SELECT statement like this, you need to give the subquery an alias like countries\_num here. Please pause the video here and carefully review this code. The result of this query comes next. [PAUSE] It's kinda like magic that this works, huh?! If you haven't discovered it already, there are often many different ways to solve problems with SQL queries. You could use a carefully constructed JOIN to achieve this same result, for example. 10. Let's practice! Test out your subquery expertise with a few exerci  **Daily XP1750** ExerciseExerciseSubquery inside where You'll now try to figure out which countries had high average life expectancies (at the country level) in 2015. Instructions 1/2 **50 XP**   * [1](javascript:void(0)) * [2](javascript:void(0))   Begin by calculating the average life expectancy across all countries for 2015.  -- Select average life\_expectancy  \_\_\_    -- From populations    \_\_\_  -- Where year is 2015  \_\_\_  -- Select average life\_expectancy  SELECT AVG(life\_expectancy)    -- From populations    FROM populations  -- Where year is 2015  WHERE year = 2015;   | **avg** | | --- | | 71.6763415481105 |   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 \*  FROM populations  WHERE life\_expectancy > 1.15 \* 100  AND year = 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.  -- Select fields  \_\_\_    -- From populations    \_\_\_  -- Where life\_expectancy is greater than  \_\_\_    -- 1.15 \* subquery    \_\_\_     \_\_\_     \_\_\_    \_\_\_;  SELECT \_\_\_  FROM \_\_\_  WHERE \_\_\_ >  \_\_\_ \* (SELECT \_\_\_(\_\_\_)  FROM \_\_\_  WHERE \_\_\_ = \_\_\_) AND  \_\_\_ = \_\_\_;  -- Select fields  SELECT \*    -- From populations  FROM populations  -- Where life\_expectancy is greater than  WHERE life\_expectancy >    -- 1.15 \* subquery   1.15 \*(SELECT AVG(life\_expectancy)     FROM populations      WHERE year = 2015) AND      year = 2015;     | **pop\_id** | **country\_code** | **year** | **fertility\_rate** | **life\_expectancy** | **size** | | --- | --- | --- | --- | --- | --- | | 21 | AUS | 2015 | 1.833 | 82.45122 | 23789752 | | 376 | CHE | 2015 | 1.54 | 83.19756 | 8281430 | | 356 | ESP | 2015 | 1.32 | 83.380486 | 46443992 | | 134 | FRA | 2015 | 2.01 | 82.67073 | 66538392 | | 170 | HKG | 2015 | 1.195 | 84.278046 | 7305700 | | 174 | ISL | 2015 | 1.93 | 82.86098 | 330815 | | 190 | ITA | 2015 | 1.37 | 83.49024 | 60730584 | | 194 | JPN | 2015 | 1.46 | 83.84366 | 126958470 | | 340 | SGP | 2015 | 1.24 | 82.59512 | 5535002 | | 374 | SWE | 2 |  |  |  |   **Daily XP2035**  **Exercise**  **Exercise**  **Subquery inside where (2)**  Use your knowledge of subqueries in WHERE to get the urban area population for only capital cities.  **Instructions**  **100 XP**   * Make use of the capital field in the countries table in your subquery. * Select the city name, country code, and urban area population fields.   -- Select fields  \_\_\_ \_\_\_, \_\_\_, urbanarea\_pop    -- From cities    \_\_\_ \_\_\_  -- Where city name in the field of capital cities  \_\_\_ \_\_\_ IN    -- Subquery    (\_\_\_ \_\_\_     \_\_\_ \_\_\_)  ORDER BY urbanarea\_pop DESC;  -- Select fields  SELECT name, country\_code, urbanarea\_pop    -- From cities    FROM cities  -- Where city name in the field of capital cities  WHERE name IN    -- Subquery    (SELECT capital     FROM countries)  ORDER BY urbanarea\_pop DESC;   | **name** | **country\_code** | **urbanarea\_pop** | | --- | --- | --- | | Beijing | CHN | 21516000 | | Dhaka | BGD | 14543124 | | Tokyo | JPN | 13513734 | | Moscow | RUS | 12197596 | | Cairo | EGY | 10230350 | | Kinshasa | COD | 10130000 | | Jakarta | IDN | 10075310 | | Seoul | KOR | 9995784 | | Mexico City | MEX | 8974724 | | Lima | PER | 8852000 | | London | GBR | 8673713 | | Bangkok | THA | 8280925 | | Tehran | IRN | 8154051 | | Bogota | COL |  |   Alright. You've got some practice on subqueries inside WHERE now. Time to see how you do when these subqueries are in the SELECT statement!  **Daily XP2135** ExerciseExerciseSubquery 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 the first query 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 second query to get the same result as the provided code. Instructions 1/2 **50 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * Submit the code to view the result of the provided query.   SELECT countries.name AS country, COUNT(\*) AS cities\_num    FROM cities      INNER JOIN countries      ON countries.code = cities.country\_code  GROUP BY country  ORDER BY cities\_num DESC, country  LIMIT 9;  /\*  SELECT \_\_\_ AS \_\_\_,    (SELECT \_\_\_     FROM \_\_\_     WHERE countries.code = cities.country\_code) AS cities\_num  FROM \_\_\_  ORDER BY \_\_\_ \_\_\_, \_\_\_  LIMIT 9;  \*/   | **country** | **cities\_num** | | --- | --- | | China | 36 | | India | 18 | | Japan | 11 | | Brazil | 10 | | Pakistan | 9 | | United States | 9 | | Indonesia | 7 | | Russian Federation | 7 | | South Korea | 7 |   /\*  SELECT countries.name AS country, COUNT(\*) AS cities\_num    FROM cities      INNER JOIN countries      ON countries.code = cities.country\_code  GROUP BY country  ORDER BY cities\_num DESC, country  LIMIT 9;  \*/  SELECT countries.name AS country,    -- Subquery    (SELECT COUNT(\*) AS cities\_num     FROM cities     WHERE countries.code = cities.country\_code) AS cities\_num  FROM countries  ORDER BY cities\_num DESC, country  LIMIT 9;   | **country** | **cities\_num** | | --- | --- | | China | 36 | | India | 18 | | Japan | 11 | | Brazil | 10 | | Pakistan | 9 | | United States | 9 | | Indonesia | 7 | | Russian Federation | 7 | | South Korea |  |   Great! The next video will introduce you to using subqueries in the FROM clause. Exciting stuff  **Daily XP2235** Subquery inside FROM clause **50 XP** 1. Subquery inside the FROM clause The last basic type of a subquery exists inside of a FROM clause. A motivating example pertaining to the percentage of women in parliament will be used now to help you understand this style of subquery. Let's dig in! 2. Build-up First, let's determine the maximum percentage of women in parliament for each continent listed in states. Recall that this query will only work if you include continent as one of the fields in the SELECT clause since we are grouping based on that field. Let's check out the result. We see that Europe has the largest value and North America has the smallest value for the countries listed in the states table. 3. Focusing on records in monarchs What if you weren't interested in all continents, but specifically those in the monarchs table. You haven't seen this yet in the course but you can include multiple tables in a FROM clause by adding a comma between them. Let's investigate a way to get the continents only in monarchs using this new trick. We have at least part of our answer here, but how do we get rid of those duplicate entries? And what about the maximum column? 4. Finishing off the subquery To get Asia and Europe to appear only once, use the DISTINCT command in your SELECT statement. But now how do you get that maximum column to also come along with Asia and Europe? Instead of including states in the FROM clause, include the subquery instead and alias it with a name like subquery. There you have it! That's how to include a subquery as a temporary table in your FROM clause. 5. Let's practice! You are very close to the end of the course. Awesome work on making it this far! The last remaining exercises are designed to really test your understanding of the material. You may be a bit frustrated but stick with it and you'll take the knowledge gained in these problems with you as you continue to develop your SQL skills. See you in a bit for the course review video!  **Daily XP50** ExerciseExerciseSubquery 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.) Instructions 1/2 **50 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * 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.   -- Select fields (with aliases)  \_\_\_    -- From languages    \_\_\_  -- Group by code  \_\_\_;  -- Select fields (with aliases)  SELECT code, COUNT(name) AS lang\_num    -- From languages    FROM languages  -- Group by code  GROUP BY code;   | **code** | **lang\_num** | | --- | --- | | PRY | 2 | | NRU | 3 | | MDG | 3 | | ASM | 5 | | TZA | 4 | | PLW | 6 | | NLD | 1 | | VEN | 2 | | BMU | 2 | | MSR | 1 | | COG | 3 | |  | | * 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.   -- Select fields  \_\_\_    -- From countries    \_\_\_      -- Subquery (alias as subquery)      (SELECT code, COUNT(\*) AS lang\_num       FROM languages       GROUP BY code) AS \_\_\_    -- Where codes match    \_\_\_  -- Order by descending number of languages  \_\_\_;  -- Select fields   | **local\_name** | **lang\_num** | | --- | --- | | Zambia | 19 | | YeItyop´iya | 16 | | Zimbabwe | 16 | | Nepal | 14 | | Bharat/India | 14 | | Mali | 13 | | South Africa | 13 | | France | 13 | | Angola | 12 | | Malawi | 12 | | Ghana | 11 | | Panama | 11 | | Pakistan |  | |  | | **Daily XP100** ExerciseExerciseAdvanced subquery You can also nest multiple subqueries to answer even more specific questions.  In this exercise, for each of the six continents listed in 2015, you'll identify which country had the maximum inflation rate, and how high it was, using multiple subqueries. The table result of your final query should look something like the following, where anything between < > will be filled in with appropriate values:  +------------+---------------+-------------------+  | name | continent | inflation\_rate |  |------------+---------------+-------------------|  | <country1> | North America | <max\_inflation1> |  | <country2> | Africa | <max\_inflation2> |  | <country3> | Oceania | <max\_inflation3> |  | <country4> | Europe | <max\_inflation4> |  | <country5> | South America | <max\_inflation5> |  | <country6> | Asia | <max\_inflation6> |  +------------+---------------+-------------------+  Again, there are multiple ways to get to this solution using only joins, but the focus here is on showing you an introduction into advanced subqueries. Instructions 1/3 **35 XP**   * [1](javascript:void(0)) * [2](javascript:void(0)) * [3](javascript:void(0)) * Create an INNER JOIN with countries on the left and economies on the right with USING, without aliasing your tables or columns. * Retrieve the country's name, continent, and inflation rate for 2015.   -- Select fields  \_\_\_    -- From countries    \_\_\_      -- Join to economies      \_\_\_      -- Match on code      \_\_\_  -- Where year is 2015  \_\_\_;  -- Select fields  SELECT name, continent, inflation\_rate    -- From countries    FROM countries      -- Join to economies      INNER JOIN economies      -- Match on code      ON countries.code = economies.code  -- Where year is 2015  WHERE year = 2015;   | **name** | **continent** | **inflation\_rate** | | --- | --- | --- | | Afghanistan | Asia | -1.549 | | Angola | Africa | 10.287 | | Albania | Europe | 1.896 | | United Arab Emirates | Asia | 4.07 | | Argentina | South America | null | | Armenia | Asia | 3.731 | | Antigua and Barbuda | North America | 0.969 | | Australia | Oceania | 1.461 | | Austria | Europe | 0.81 | | Azerbaijan | Asia | 4.049 | | Burundi | Africa | 5.553 | | Belgium | Europe | 0.62 | | Benin |  |  |   Select the maximum inflation rate in 2015 AS max\_inf grouped by continent using the previous step's query as a subquery in the FROM clause.   * Thus, in your subquery you should:   + Create an inner join with countries on the left and economies on the right with USING (without aliasing your tables or columns).   + Retrieve the country name, continent, and inflation rate for 2015.   + Alias the subquery as subquery.   This will result in the six maximum inflation rates in 2015 for the six continents as one field table. Make sure to not include continent in the outer SELECT statement.  -- Select the maximum inflation rate as max\_inf  \_\_\_    -- Subquery using FROM (alias as subquery)    FROM (        SELECT name, continent, inflation\_rate        FROM countries        INNER JOIN economies        USING (code)        WHERE year = 2015) AS \_\_\_  -- Group by continent  \_\_\_;  -- Select the maximum inflation rate as max\_inf  SELECT MAX(inflation\_rate) AS max\_inf    -- Subquery using FROM (alias as subquery)    FROM (        SELECT name, continent, inflation\_rate        FROM countries        INNER JOIN economies        USING (code)        WHERE year = 2015) AS subquery  -- Group by continent  GROUP BY continent;   | **max\_inf** | | --- | | 21.858 | | 39.403 | | 121.738 | | 7.524 | | 48.684 | | 9.784 |  * Now it's time to append your second query to your first query using AND and IN to obtain the name of the country, its continent, and the maximum inflation rate for each continent in 2015. * For the sake of practice, change all joining conditions to use ON instead of USING.   -- Select fields  SELECT name, continent, inflation\_rate    -- From countries    FROM countries    -- Join to economies    INNER JOIN economies    -- Match on code    \_\_\_    -- Where year is 2015    WHERE year = 2015      -- And inflation rate in subquery (alias as subquery)      \_\_\_ \_\_\_ \_\_\_ (          SELECT MAX(inflation\_rate) AS max\_inf          FROM (               SELECT name, continent, inflation\_rate               FROM countries               INNER JOIN economies               \_\_\_               WHERE year = 2015) AS subquery        -- Group by continent          GROUP BY continent);  -- Select fields  SELECT name, continent, inflation\_rate    -- From countries    FROM countries    -- Join to economies    INNER JOIN economies    -- Match on code    ON countries.code = economies.code    -- Where year is 2015    WHERE year = 2015      -- And inflation rate in subquery (alias as subquery)      AND inflation\_rate IN (          SELECT MAX(inflation\_rate) AS max\_inf          FROM (               SELECT name, continent, inflation\_rate               FROM countries               INNER JOIN economies               ON countries.code = economies.code               WHERE year = 2015) AS subquery        -- Group by continent          GROUP BY continent);   | **name** | **continent** | **inflation\_rate** | | --- | --- | --- | | Haiti | North America | 7.524 | | Malawi | Africa | 21.858 | | Nauru | Oceania | 9.784 | | Ukraine | Europe | 48.684 | | Venezuela | South America | 121.738 | | Yemen |  |  |   Wow! Well done! This code works since each of the six maximum inflation rate values occur only once in the 2015 data. Think about whether this particular code involving subqueries would work in cases where there are ties for the maximum inflation rate values.  **Daily XP200**  **Exercise**  **Exercise**  **Subquery challenge**  Let's test your understanding of the subqueries with a challenge problem! Use a subquery to get 2015 economic data for countries that do **not** have   * gov\_form of 'Constitutional Monarchy' or * 'Republic' in their gov\_form.   Here, gov\_form stands for the form of the government for each country. Review the different entries for gov\_form in the countries table.  **Instructions**  **100 XP**   * Select the country code, inflation rate, and unemployment rate. * Order by inflation rate ascending. * Do not use table aliasing in this exercise.   -- Select fields  SELECT \_\_\_, \_\_\_, \_\_\_    -- From economies    FROM \_\_\_    -- Where year is 2015 and code is not in    WHERE \_\_\_ = \_\_\_ AND \_\_\_ \_\_\_ \_\_\_      -- Subquery      (SELECT \_\_\_       FROM \_\_\_       WHERE (\_\_\_ = \_\_\_ OR \_\_\_ LIKE \_\_\_))  -- Order by inflation rate  ORDER BY \_\_\_;  -- Select fields  SELECT code, inflation\_rate, unemployment\_rate    -- From economies    FROM economies    -- Where year is 2015 and code is not in    WHERE year = 2015 AND code NOT IN      -- Subquery      (SELECT code       FROM countries       WHERE (gov\_form = 'Constitutional Monarchy' OR gov\_form LIKE '%Republic%'))  -- Order by inflation rate  ORDER BY inflation\_rate;   | **code** | **inflation\_rate** | **unemployment\_rate** | | --- | --- | --- | | AFG | -1.549 | null | | CHE | -1.14 | 3.178 | | PRI | -0.751 | 12 | | ROU | -0.596 | 6.812 | | BRN | -0.423 | 6.9 | | TON | -0.283 | null | | OMN | 0.065 | null | | TLS | 0.553 | null | | BEL | 0.62 | 8.492 | | CAN | 1.132 | 6.9 | | MNE | 1.204 | null | | SRB | 1.392 | 18.2 | | AUS | 1.461 | 6.058 | | QAT | 1.814 | null | | BHR | 1.836 | null | | WSM | 1.923 | null | | MYS | 2.104 | 3.1 | | SAU | 2.189 | 5.591 | | HKG | 3.037 | 3.296 | | KWT | 3.233 | 2.072 | | ARE | 4.07 | null | | MAC | 4.564 | 1.825 | | SWZ | 4.96 | null | | BTN | 6.336 |  |   Superb! Let's review subqueries before you head off to the last video of this course!  **Subquery review**  Within which SQL clause are subqueries most frequently found?  **Answer the question**  **50XP**  **Possible Answers**      **WHERE**  press1      FROM  press2      SELECT  press3      IN  press4 |  | | **Daily XP350** Course review **50 XP** 1. Course Review Only the challenge problems remain! Way to go! You're on your way to being a SQL query warrior using PostgreSQL! Before you tackle the three challenge problems, let's review the main topics covered throughout the course. 2. Types of joins In SQL, a join combines columns from one or more tables in a relational database via a lookup process. There are four different types of joins you learned about in this course. First, an INNER JOIN is also denoted as just JOIN in SQL. A special case of an INNER JOIN you explored is called a self-join. Second, there are three OUTER JOINs denoted as LEFT JOIN (or LEFT OUTER JOIN), RIGHT JOIN (or RIGHT OUTER JOIN), and FULL JOIN (or FULL OUTER JOIN). Third, you worked with CROSS JOINs to create all possible combinations between two tables. Lastly, you investigated semi-joins and anti-joins. Remember that words appearing in ALL capital letters correspond to the joins having simple SQL syntax. Self-joins, semi-joins, and anti-joins don't have built-in SQL syntax. 3. INNER JOIN vs LEFT JOIN An INNER JOIN keeps only the records in which the key field (or fields) is in both tables. A LEFT JOIN keeps all the records in fields specified in the left table and includes the matches in the right table based on the key field or fields. Key field values that don't match in the right table are included as missing data in the resulting table of a LEFT JOIN. 4. RIGHT JOIN vs FULL JOIN A RIGHT JOIN keeps all the records specified in the right table and includes the matches from the key field (or fields) in the left table. Those that don't match are included as missing values in the resulting table from the RIGHT JOIN query. A FULL JOIN is a combination of a LEFT JOIN and a RIGHT JOIN showing exactly which values appear in both tables and those that appear in only one or the other table. 5. CROSS JOIN with code A CROSS JOIN matches all records from fields specified in one table with all records from fields specified in another table. Remember that a CROSS JOIN does not have an ON or USING clause, but otherwise looks very similar to the code for an INNER JOIN, LEFT JOIN, RIGHT JOIN, or FULL JOIN. 6. Set Theory Clauses Recall that UNION includes every record in both tables but DOES NOT double count those that are in both tables whereas UNION ALL DOES replicate those that are in both tables. INTERSECT gives only those records found in both of the two tables. EXCEPT gives only those records in one table BUT NOT the other. 7. Semi-joins and Anti-joins When you'd like to filter your first table based on conditions set on a second table, you should use a semi-join to accomplish your task. If instead you'd like to filter your first table based on conditions NOT being met on a second table, you should use an anti-join. Anti-joins are particularly useful in diagnosing problems with other joins in terms of getting fewer or more records than you expected. 8. Types of basic subqueries The most common type of subquery is done inside of a WHERE clause. The next most frequent types of subqueries are inside SELECT clauses and inside FROM clauses. As you'll see in the challenge exercises, subqueries can also find their way into the ON statement of a join in ways similar to what you've seen inside WHERE clauses too. 9. Own the challenge problems! You got this! Well, you are only three exercises away from mastering the content in this course. You are a true SQL ninja. Now take down these last three problems using all the skills you've built up in this course!  **Daily XP400**  **Exercise**  **Exercise**  **Final challenge**  Welcome to the end of the course! The next three exercises will test your knowledge of the content covered in this course and apply many of the ideas you've seen to difficult problems. Good luck!  Read carefully over the instructions and solve them step-by-step, thinking about how the different clauses work together.  In this exercise, you'll need to get the country names and other 2015 data in the economies table and the countries table for **Central American countries with an official language**.  **Instructions**  **100 XP**   * Select unique country names. Also select the total investment and imports fields. * Use a left join with countries on the left. (An inner join would also work, but please use a left join here.) * Match on code in the two tables AND use a subquery inside of ON to choose the appropriate languages records. * Order by country name ascending. * Use table aliasing but **not** field aliasing in this exercise.   -- Select fields  SELECT DISTINCT \_\_\_, \_\_\_, \_\_\_    -- From table (with alias)    FROM \_\_\_ AS \_\_\_      -- Join with table (with alias)      LEFT JOIN \_\_\_ AS \_\_\_        -- Match on code        ON (\_\_\_.\_\_\_ = \_\_\_.\_\_\_        -- and code in Subquery          AND \_\_\_ IN (            SELECT \_\_\_.\_\_\_            FROM \_\_\_ AS \_\_\_            WHERE \_\_\_ = '\_\_\_'          ) )    -- Where region and year are correct    WHERE \_\_\_ = \_\_\_ AND \_\_\_ = \_\_\_  -- Order by field  ORDER BY \_\_\_;  -- Select fields  SELECT DISTINCT(c.name), total\_investment, imports    -- From table (with alias)    FROM countries AS c      -- Join with table (with alias)      LEFT JOIN economies AS e        -- Match on code        ON (c.code = e.code        -- and code in Subquery          AND c.code IN (            SELECT l.code            FROM languages AS l            WHERE official = 'true'          ) )    -- Where region and year are correct    WHERE region = 'Central America' AND year = 2015  -- Order by field  ORDER BY c.name;   | **ame** | **total\_investment** | **imports** | | --- | --- | --- | | Belize | 22.014 | 6.743 | | Costa Rica | 20.218 | 4.629 | | El Salvador | 13.983 | 8.193 | | Guatemala | 13.433 | 15.124 | | Honduras | 24.633 | 9.353 | | Nicaragua | 31.862 | 11.665 | | Panama |  |  |   **Daily XP500**  **Exercise**  **Exercise**  **Final challenge (2)**  Whoofta! That was challenging, huh?  Let's ease up a bit and calculate the average fertility rate for each region in 2015.  **Instructions**  **100 XP**   * Include the name of region, its continent, and average fertility rate aliased as avg\_fert\_rate. * Sort based on avg\_fert\_rate ascending. * Remember that you'll need to GROUP BY all fields that aren't included in the aggregate function of SELECT.   -- Select fields  SELECT \_\_\_, \_\_\_, \_\_\_(\_\_\_) AS \_\_\_    -- From left table    FROM \_\_\_ AS \_\_\_      -- Join to right table      INNER JOIN \_\_\_ AS \_\_\_        -- Match on join condition        ON \_\_\_ = \_\_\_    -- Where specific records matching some condition    WHERE \_\_\_ = \_\_\_  -- Group appropriately  GROUP BY \_\_\_, \_\_\_  -- Order appropriately  ORDER BY \_\_\_;  -- Select fields  SELECT region, continent, AVG(fertility\_rate) AS avg\_fert\_rate    -- From left table    FROM countries AS c      -- Join to right table      INNER JOIN populations AS p        -- Match on join condition        ON c.code = p.country\_code    -- Where specific records matching some condition    WHERE year = 2015  -- Group appropriately  GROUP BY region, continent  -- Order appropriately  ORDER BY avg\_fert\_rate;   | **region** | **continent** | **avg\_fert\_rate** | | --- | --- | --- | | Southern Europe | Europe | 1.4261000037193299 | | Eastern Europe | Europe | 1.490888900227017 | | Baltic Countries | Europe | 1.603333314259847 | | Western Europe | Europe | 1.6325000077486038 | | Eastern Asia | Asia | 1.6916666825612385 | | North America | North America | 1.7657500207424164 | | British Islands | Europe | 1.875 | | Nordic Countries | Europe | 1.8933333555857341 | | Australia and New Zealand | Oceania | 1.9114999771118164 | | Caribbean | North America | 1.9505714348384313 | | Southeast Asia | Asia | 2.156000018119812 | | South America | South America | 2.274750014146169 | | Central America | North America | 2.3263749927282333 | | Middle East | Asia | 2.547055568959978 | | Southern and Central Asia | Asia | 2.6341428586414883 | | Micronesia | Oceania | 2.864750027656555 | | Northern Africa | Africa | 2.9081666469573975 | | Southern Africa |  |  |   Interesting. It seems that the average fertility rate is lowest in Southern Europe and highest in Central Africa. Two down, one to go!  **Daily XP650**  **Exercise**  **Exercise**  **Final challenge (3)**  Welcome to the last challenge problem. By now you're a query warrior! Remember that these challenges are designed to take you to the limit to solidify your SQL knowledge! Take a deep breath and solve this step-by-step.  You are now tasked with determining the top 10 capital cities in Europe and the Americas in terms of a calculated percentage using city\_proper\_pop and metroarea\_pop in cities.  Do not use table aliasing in this exercise.  **Instructions**  **100 XP**   * Select the city name, country code, city proper population, and metro area population. * Calculate the percentage of metro area population composed of city proper population for each city in cities, aliased as city\_perc. * Focus only on capital cities in Europe and the Americas in a subquery. * Make sure to exclude records with missing data on metro area population. * Order the result by city\_perc descending. * Then determine the top 10 capital cities in Europe and the Americas in terms of this city\_perc percentage.   -- Select fields  SELECT \_\_\_, \_\_\_, \_\_\_, \_\_\_,        -- Calculate city\_perc        \_\_\_ / \_\_\_ \* \_\_\_ AS \_\_\_    -- From appropriate table    FROM \_\_\_    -- Where    WHERE \_\_\_ IN      -- Subquery      (SELECT capital       FROM \_\_\_       WHERE (\_\_\_ = \_\_\_          OR \_\_\_ LIKE \_\_\_))         AND \_\_\_ IS \_\_\_ \_\_\_  -- Order appropriately  ORDER BY \_\_\_ \_\_\_  -- Limit amount  \_\_\_ \_\_\_;  -- Select fields  SELECT name, country\_code, city\_proper\_pop, metroarea\_pop,        -- Calculate city\_perc        city\_proper\_pop / metroarea\_pop \* 100 AS city\_perc    -- From appropriate table    FROM cities    -- Where    WHERE name IN      -- Subquery      (SELECT capital       FROM countries       WHERE (continent = 'Europe'          OR continent LIKE '%America'))         AND cities.metroarea\_pop IS NOT NULL  -- Order appropriately  ORDER BY city\_perc DESC  -- Limit amount  LIMIT 10;   | **name** | **country\_code** | **city\_proper\_pop** | **metroarea\_pop** | **city\_perc** | | --- | --- | --- | --- | --- | | Lima | PER | 8852000 | 10750000 | 82.34418630599976 | | Bogota | COL | 7878783 | 9800000 | 80.3957462310791 | | Moscow | RUS | 12197596 | 16170000 | 75.43349266052246 | | Vienna | AUT | 1863881 | 2600000 | 71.6877281665802 | | Montevideo | URY | 1305082 | 1947604 | 67.00961589813232 | | Caracas | VEN | 1943901 | 2923959 | 66.48181676864624 | | Rome | ITA | 2877215 | 4353775 | 66.0855233669281 | | Brasilia | BRA | 2556149 | 3919864 | 65.2101457118988 | | London | GBR | 8673713 | 13879757 | 62.491822242736816 | | Budapest | HUN | 1759407 | 2927944 | 60.09018421173096 |   That's a wrap! Check out the excellent follow-up course entitled [Intermediate SQL](https://www.datacamp.com/courses/intermediate-sql) by Mona Khalil too! |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |