Reference

This is a Datacamp course

Set theory clauses

In this chapter, you'll learn more about set theory using Venn diagrams and you will be introduced to union, union all, intersect, and except clauses. You'll finish by investigating semi-joins and anti-joins, which provide a nice introduction to subqueries.

Union does not double counting, Union all double counting. Except only show the results of one table but not other.

Union

Near query result to the right, you will see two new tables with names economies2010 and economies2015.

Combine these two tables into one table containing all of the fields in economies2010. The economies table is also included for reference.
Sort this resulting single table by country code and then by year, both in ascending order.

select
from economies2010
UNION ALL
select
from economies2015
order by code, year;

It just stack them together.

Note two tables must have same columns to achieve Union, Union ALL, Intersect, etc.?

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 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 country_code
FROM cities
UNION
SELECT code
FROM currencies
ORDER BY country_code;
```

Note the word non-duplicated. Thus we have used UNION instead of UNION ALL.

Union all

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

To include duplicates, you can use UNION ALL.

Determine all combinations (include duplicates) of country code and year that exist in either the economies or the populations tables.
Order by code then year.
The result of the query should only have two columns/fields.
Think about how many records this query should result in.
You'll use code very similar to this in your next exercise after the video. Make note of this code after completing it.

```
SELECT code, year
FROM economies
UNION ALL
SELECT country_code, year
FROM populations
ORDER BY code, year;
```
Order by code but not country_code.

Intersect

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

Again, order by code and then by year, both in ascending order. Note the number of records here (given at the bottom of query result) compared to the similar UNION ALL query result (814 records).

```
SELECT code, year
FROM economies
INTERSECT SELECT country_code, year
FROM populations
ORDER BY code, year;
```

Intersect (2)

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

Use INTERSECT to answer this question with countries and cities!

```
select name
from countries
INTERSECT
select name
from cities
```

Except

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

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

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

```
SELECT name
FROM cities
EXCEPT
```

```
SELECT capital
FROM countries
ORDER BY name;
```

Except (2)

Now you will complete the previous query in reverse!

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

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

```
select capital
from countries
EXCEPT
select name
from cities
order by capital
```

Semi-join

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

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

```
select code
from countries
where region = 'Middle East'
```

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

Comment out the answer to the previous tab by surrounding it in / *and* /. You'll come back to it!
Below the commented code, select only unique languages by name appearing in the languages table.
Order the resulting single field table by name in ascending order.

```
select distinct name
from languages
order by name
```

Now combine the previous two queries into one query using WHERE to determine the unique languages spoken in the Middle East. Carefully review this result and its code after completing it. It serves as a great example of subqueries, which are the focus of Chapter 4.

```
select distinct name
from languages
where code in
(
select code
from countries
where region = 'Middle East'
)
order by name
```

Relating semi-join to a tweaked inner join

Let's revisit the code from the previous exercise. Sometimes problems solved with semi-joins can also be solved using an inner join.

What is missing from the code at the bottom of the editor to get it to match with the correct answer produced by the commented out code at the top of the editor, which retrieves languages spoken in the Middle East?

Possible Answers [1] HAVING instead of WHERE [2] DISTINCT [3] UNIQUE

```
-- Previous exercise
/
SELECT DISTINCT name
FROM languages
WHERE code IN
(SELECT code
FROM countries
WHERE region = 'Middle East')
ORDER BY name; /
```

```
SELECT languages.name AS language
FROM languages
INNER JOIN countries
ON languages.code = countries.code
WHERE region = 'Middle East'
ORDER BY language;
```

Note the answer is distinct. This is either inner join or outer join might produce duplicate results.

Diagnosing problems using anti-join

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

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

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

```
select count(*)
from countries
where continent = 'Oceania'
```

- Complete an inner join with countries AS c1 on the left and currencies AS c2 on the right to get the different currencies used in the countries of Oceania.
- Match ON the code field in the two tables.
- Include the country code, country name, and basic_unit AS currency.
  Observe query result and make note of how many different countries are listed here.

```
select c1.code, c1.name,c2.basic_unit AS currency
from countries AS c1
INNER JOIN currencies AS c2
ON c1.code = c2.code
where continent = 'Oceania'
```

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 code, name
FROM countries
WHERE continent = 'Oceania'
AND code NOT IN
(SELECT code
FROM currencies);
```

**NOT IN, or IN is in not

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!

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 name
from cities AS c1
WHERE c1.country_code IN
(
SELECT e.code
FROM economies AS e
UNION
SELECT c2.code
FROM currencies AS c2
-- exclude those appearing in populations AS p
EXCEPT
SELECT p.country_code
FROM populations AS p
);
```

Subqueries (or nested queries)

In this closing chapter, you'll learn how to use nested queries to add some finesse to your data insights. You'll also wrap all of the content covered throughout this course into solving three challenge problems.

Subquery inside where

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

Begin by calculating the average life expectancy across all countries for 2015.

```
select avg(life_expectancy)
from populations
where year = 2015
```

Recall that you can use SQL to do calculations for you. Suppose we wanted only records that were below 0.5 * 100 in terms of life expectancy for 2010:

SELECT *FROM populations WHERE life_expectancy < 0.5* 100 AND year = 2010;

Select all fields from populations with records corresponding to larger than 1.15 times the average you calculated in the first task for 2015.

You should use the query in the last task as a subquery here instead of the hard-coding used in the example text above.

```
SELECT
FROM populations
WHERE life_expectancy >
1.15 (SELECT AVG(life_expectancy)
FROM populations
WHERE year = 2015) AND
year = 2015;
```

Note we need year = 2015 again. Be careful! The outer select statement cannot see the 2015 used in the inner query.

Subquery inside where (2)

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

Make use of the capital field in the countries table in your subquery. Select the city name, country code, and urban area population fields.

```
select name, country_code, urbanarea_pop
from cities
where name IN
(select capital
from countries)
ORDER BY urbanarea_pop DESC;
```

Subquery inside select

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

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

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

SELECT countries.name AS country, COUNT*() AS cities_num **Here count()* and count(cities.name) give same results
FROM cities
INNER JOIN countries
ON countries.code = cities.country_code
GROUP BY country
ORDER BY cities_num DESC, country
LIMIT 9; Here order descending by cities_num first and country second. Nothing to do with the order of columns in select statement**.

- remove the comments around the second query and comment out the first query instead.
- Convert the GROUP BY code to use a subquery inside of SELECT, i.e. fill in the blanks to get a result that matches the one given using the GROUP BY code in the first query.
- Again, sort the result by cities_num descending and then by country ascending.

```
SELECT countries.name AS country,
(SELECT COUNT(*)
FROM cities
WHERE countries.code = cities.country_code) AS cities_num
```

```
FROM countries
ORDER BY cities_num DESC, country
LIMIT 9;
```

In the group by case, we are handling a join table and thus we have information citi_num. However, in the subquery case, we are selecting from countries. How to imagine we the information of cities_num? It can be there even if it is not in the countries table, as long as we can obtain it from elsewhere. In fact, we can select a lot of things that is not in the table name after from, right?

Subquery inside from

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

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

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 code, count(*) as lang_num
from languages
group by code
```

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 local_name, subquery.lang_num
FROM countries,
(SELECT code, COUNT(*) AS lang_num
FROM languages
GROUP BY code) AS subquery
WHERE countries.code = subquery.code
ORDER BY lang_num DESC;
```

The above solved with a join. The key is to understand that I need a table with local_name and the corresponding languages, and then use group by. The following seems more evident.

```
select countries.local_name, count(*) as lang_num
from countries
INNER JOIN languages
ON countries.code = languages.code
group by countries.local_name
```

Advanced 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 query in Task 3 should look something like the following, where anything between < > will be filled in with appropriate values:

```
+------------+---------------+------------------+
| name | continent | inflation_rate |
|------------+---------------+------------------|
| | North America | |
| | Africa | |
| | Oceania | |
| | Europe | |
| | South America | |
| | Asia | |
+------------+---------------+------------------+
```

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.

Create an inner join with countries on the left and economies on the right with USING. Do not alias your tables or columns.

Retrieve the country name, continent, and inflation rate for 2015.

```
SELECT name, continent, inflation_rate
FROM countries
INNER JOIN economies
```

```
USING (code)
WHERE year = 2015;
```
In my version, I did alias. It is actually unnecessary because no same column names

- Determine the maximum inflation rate for each continent in 2015 using the previous query as a subquery called subquery in the FROM clause.
- Select the maximum inflation rate AS max_inf grouped by continent.

This will result in the six maximum inflation rates in 2015 for the six continents as one field table. (Don't include continent in the outer SELECT statement.)

```
SELECT MAX(inflation_rate) AS max_inf
FROM (
SELECT name, continent, inflation_rate
FROM countries
INNER JOIN economies
USING (code)
WHERE year = 2015) AS subquery
GROUP BY continent;
```

NOTE, here the continent in GROUP BY is not appeared after SELECT

- Append the second part's query to the first part's query using WHERE, AND, and IN to obtain the name of the country, its continent, and the maximum inflation rate for each continent in 2015. Revisit the sample output in the assignment text at the beginning of the exercise to see how this matches up.
- For the sake of practice, change all joining conditions to use ON instead of USING.

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.

```
SELECT name, continent, inflation_rate
FROM countries
INNER JOIN economies
ON countries.code = economies.code
WHERE year = 2015
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);

My version-I SELECT name, continent, max(inflation_rate) AS MaxInflation
FROM countries
INNER JOIN economies
USING (code)
WHERE year = 2015
GROUP BY continent,name

This is obviously wrong. When grouping by name and continent, we have only one record for year 2015 for each country. Thus we can calculate max within that record. It is not the maximum inflation rate within all the countries in a continent.

Now I changed it to:
SELECT continent, max(inflation_rate) AS MaxInflation
FROM countries
INNER JOIN economies
USING (code)
WHERE year = 2015
GROUP BY continent

This give a two column table. This is normally required in many problems. However, it is not what we required in this problem, as shown in the table earlier. In other words, we miss a country-name column.

My version_II.
As earlier, we know that we can have a two-column table (continent and maxInflation_rate), just missing the required country_Name column. A natural idea is to construct another table and that also have these two columns but also have the country_name. Then we join these two table to obtain the final results.

SELECT t1.name AS name, t1.continent AS continent, t2.MaxInflation AS inflation_rate
FROM
(
SELECT continent, name, inflation_rate
FROM countries
INNER JOIN economies

```
USING (code)
WHERE year = 2015
) as t1
INNER JOIN
(
SELECT continent, max(inflation_rate) AS MaxInflation
FROM countries
INNER JOIN economies
USING (code)
WHERE year = 2015
GROUP BY continent
) as t2
ON (t1.continent = t2.continent AND t1.inflation_rate = t2.MaxInflation)
```

SUMMARY For the problem of printing two-columns plus an aggregate quantity, we normally cannot use the typical way using GROUP BY approach for the problem of printing one-column plus an aggregate quantity. For now, I have two ways to handle this type of problem. (1) General structure with Joins: First obtain a one-column plus an aggregation quantity using the typical way, and then construct another similar table but with the extra columns needed. Finally join these two tables, usually on all the conditions. (see my version_II).
(2) General structure without Joins: Directly to select column1, columns..., **column corresponding to the required aggregate quantity (CCRAQ)** from ..... We then use subqueries and make sure that the CCRAQ is IN a condition-satisfied selected set (e.g. maximum, minimum something). Note although the general structure has no joins, the substructure may still contain joins.

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 Select the country code, inflation rate, and unemployment rate. Order by inflation rate ascending. Do not use table aliasing in this exercise.

Need print out: country code, inflation rate, and unemployment rate.

```
SELECT code, inflation_rate, unemployment_rate
FROM economies
WHERE year = 2015 AND code NOT IN
(SELECT code
FROM countries
WHERE (gov_form = 'Constitutional Monarchy' OR gov_form LIKE '%Republic%'))
ORDER BY inflation_rate;
```

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 (total_investment, imports) in the economies table and the countries table for Central American countries with an official language.

INSTRUCTIONS

- 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 distinct c.name, e.total_investment, e.imports
from countries AS c
left join economies AS e
ON( c.code = e.code
AND c.name IN
( select c1.name
from countries AS c1
INNER JOIN
languages AS l
ON c1.code = l.code
WHERE l.official = true
```

```
)
)
WHERE e.year = 2015 AND c.region = 'Central America'

order by c.name
```

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. Need print out: region, continent, average(fertility_rate).

INSTRUCTIONS

- 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 region, continent, avg(fertility_rate) as avg_fert_rate
from countries as c
INNER JOIN populations as p
ON c.code = p.country_code
WHERE p.year = 2015
Group by region, continent
order by avg_fert_rate;
```

Compare this problem with the one in (Advanced subquery), here is much simpler. See summary else where.

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. Need print out: city name, country code, city proper population, and metro area population, and the percentage.

INSTRUCTIONS

- 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. Do not use table aliasing in this exercise.

```
SELECT name, country_code, city_proper_pop, metroarea_pop,
city_proper_pop / metroarea_pop * 100 AS city_perc
FROM cities
WHERE name IN
(SELECT capital
FROM countries
WHERE (continent = 'Europe'
OR continent LIKE '%America'))
AND metroarea_pop IS NOT NULL
ORDER BY city_perc DESC
LIMIT 10;
```