Data Camp

Friday, April 14, 2017 13:17

WHERE and OR

The following query selects all films that were released in 1994 or 1995 which had a rating PG or R.

```
SELECT title
FROM films
WHERE (release_year = 1994 OR release_year = 1995)
AND (certification = 'PG' OR certification = 'R');
```

EX:

1) Get the title and release year for films released in the 90s.
2) Build on your query to filter the records to only include French or Spanish language filmss
3) Finally, restrict the query to only return films that took in more than $2M gross

```
SELECT title, release_year
FROM films
WHERE release_year >1989
AND release_year < 2000
AND (language = 'French' OR language = 'Spanish')
AND gross > 2000000;
```

Between: Checking the range

1) Get the title and release year of all films released between 1990 and 2000 (inclusive).
2) Build on your query to select only films that have budgets over $100 million
3) Only return Spanish language films
4) Include all Spanish or French films with same criteria as before

```
SELECT title, release_year
FROM films
```

FROM films
WHERE (release_year BETWEEN 1994 AND 2000)
AND budget > 100000000
AND (language = 'Spanish' OR language = 'French');

WHERE IN : Filter in many conditions. IN operator allows you to specify multiple values in a WHERE clause, making it easier and quicker to specify multiple OR conditions!

Original:
SELECT name
FROM kids
WHERE age = 2
OR age = 4
OR age = 6;

WHERE IN:
SELECT name
FROM kids
WHERE age IN (2,4,6);

LIKE and NOT LIKE: LIKE operator can be used in a WHERE clause to search for *a pattern* in a column.

There are two wildcards you can use with LIKE:

1) The % wildcard will match zero, one, or many characters in text.
2) The _ wildcard will match a single character.

Aggregate Functions: AVG, MAX, MIN, SUM

A note on arithmetic: In addition to using aggregate functions, you can perform basic arithmetic with symbols like +, -, *, and /

Note: the following gives a result of 1
SELECT (4/3);

WHY?
SQL assumes that if you divide an integer by an integer, you want to get an integer back. So be careful when dividing! If you want more precision when dividing, you can add decimal places to your numbers. For example, SELECT (4.0 / 3.0) AS result; gives you 1.3333

It's AS simple AS aliasing

Get the title and net profit (the amount a film grossed, minus its budget) for all films.

```
SELECT title, (gross - budget) AS net_profit
FROM films;
```

EX: Get the average duration in hours for all films. Aliased as avg_duration_hours

```
SELECT AVG(duration/60.0) AS avg_duration_hours
FROM films;
```

Ex:

1) Get the percentage of people who are no longer alive. Alias the result as percentage_dead. Remember to use 100.0 and not 100!

   ```
   SELECT COUNT(deathdate)*100.0/COUNT(*) AS percentage_dead
   FROM people;
   ```

2) Get the number of years between the newest film and oldest film. Alias the result as difference.

   ```
   SELECT MAX(release_year) - MIN(release_year) AS difference
   FROM films;
   ```

3) Get the number of decades the films table covers. Alias the result as number_of_decades. The top half of your fraction should be enclosed in parentheses.

```
SELECT (MAX(release_year) - MIN(release_year))/10 AS number_of_decades
FROM films;
```

ORDER BY: Sort a column of text Alphabetically (A-Z)

Sort single columns:

1) Get all details for all films except those released in 2015 and order them by duration

```
SELECT *
FROM films
WHERE release_year <> 2015
ORDER BY duration;
```

Sort single columns (DESC):

1) Get the IMDB score and film ID for every film from the reviews table, sorted from highest to lowest score.

```
SELECT imdb_score, film_id
FROM reviews
ORDER BY imdb_score DESC;
```

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

```
SELECT birthdate, name
FROM people
ORDER BY birthdate, name;
```

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

GROUP BY: Commonly, GROUP BY is used with aggregate functions like COUNT, MAX.

SELECT…FROM…WHERE…GROUP (BY)…HAVING…ORDER (BY)…;

```
SELECT release_year, AVG(budget) AS avg_budget, AVG(gross) AS avg_gross
FROM films
WHERE release_year > 1990
GROUP BY release_year
HAVING AVG(budget) > 60000000
ORDER BY AVG(gross);
```

Ex: Get the country, release year, and lowest amount grossed per release year per country. Order your results by country and release year.

```
SELECT country, release_year, MIN(gross)
FROM films
GROUP BY release_year, country
ORDER BY country, release_year;
```

HAVING: In SQL, aggregate functions can't be used in WHERE clauses. You can't use WHERE COUNT(title) > 10. You should use HAVING instead.

```
SELECT release_year
FROM films
GROUP BY release_year
HAVING COUNT(title)>10;
```

INNER JOIN : only return the rows that are in both tables

```
SELECT *
FROM left_table
INNER JOIN right_table
ON left_table.id = right_table.id
INNER JOIN new_table
ON left_table.id = new_table.id;
```

```
SELECT c.code, c.name, c.region, p.year, p.fertility_rate, e.unemployment_rate
FROM countries AS c
INNER JOIN populations AS p
ON c.code = p.code
INNER JOIN economies AS e
ON c.code = e.code AND e.year = p.year
```

Note: When you join multiple tables, be careful of repetitive variables. If so, you have to point it out.

INNER JOIN via USING：When joining tables with a common field name (e.g. countries.code = cities.code), you can use USING(code) instead.

```
SELECT c.name AS country, continent, l.name AS language, official
FROM countries AS c
INNER JOIN language AS l
USING (code);
```

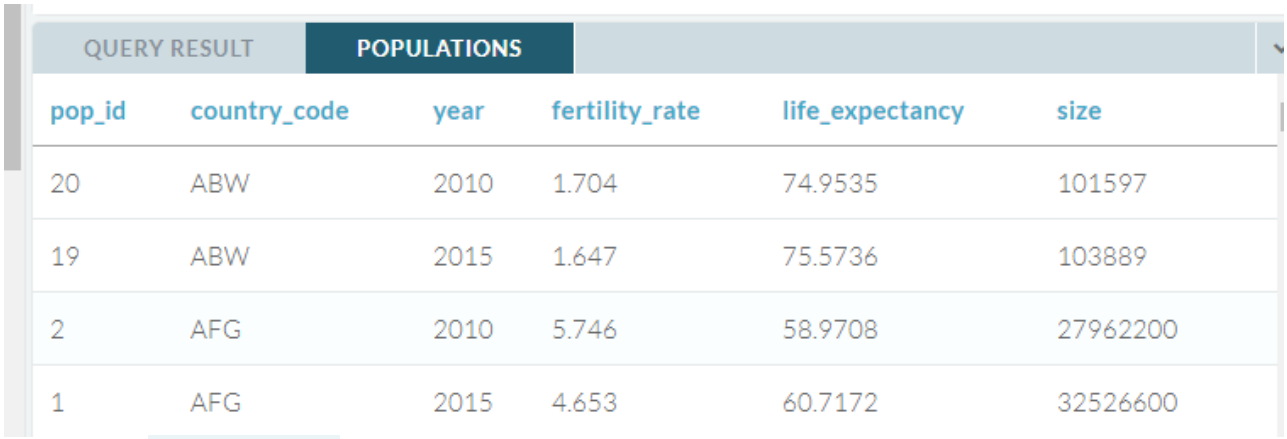
Self-ish joins, just in CASE: SELF JOIN is used to compare values in the field to the other values in the same field within the same table.

```
SELECT p1.country AS country1, p2.country AS country2, p1.continent
FROM prime_ministers AS p1
INNER JOIN prime_ministers AS p2
ON p1.continent = p2.continent AND p1.country <> p2.country
LIMIT 13;
```

CASE WHEN and THEN: to do multiple if and then else statement in a simply way in SQL

```
SELECT name, continent, indep_year
        CASE WHEN indep_year < 1990 THEN 'before 1990'
                WHEN indep_year <= 1930 THEN 'between 1990 and 1930'
                ELSE 'after 1930'
                END AS indep_year_group
FROM states
ORDER BY indep_year_group
```

Ex :



QUERY RESULT | POPULATIONS

pop_id	country_code	year	fertility_rate	life_expectancy	size
20	ABW	2010	1.704	74.9535	101597
19	ABW	2015	1.647	75.5736	103889
2	AFG	2010	5.746	58.9708	27962200
1	AFG	2015	4.653	60.7172	32526600

Use the `populations` table to perform a self-join to calculate the percentage increase in population from 2010 to 2015 for each country code!

```
SELECT p1.country_code,p1.size AS size2010,p2.size AS size2015
FROM populations AS p1
```

FROM populations AS p1
INNER JOIN populations AS p2
ON p1.country_code = p2.country_code;

country_code	size2010	size2015
ABW	101597	103889
ABW	101597	101597
ABW	103889	103889
ABW	103889	101597

For this code, you will have 4 entries laying out all combinations of 2010 and 2015

```
SELECT p1.country_code,p1.size AS size2010,p2.size AS size2015
FROM populations AS p1
INNER JOIN populations AS p2
ON p1.country_code = p2.country_code AND p1.year = p2.year-5;
```

country_code	size2010	size2015
ABW	101597	103889
AFG	27962200	32526600
AGO	21220000	25022000

Inner chanllenge

```
SELECT country_code, size,
   CASE WHEN size > 50000000
   THEN 'large'
   WHEN size > 1000000
   THEN 'medium'
   ELSE 'small' END
    AS popsize_group
-- which table?
```

```
  which table?
INTO pop_plus
FROM populations
-- any conditions to check?
WHERE populations.year = 2015;

SELECT c.name, c.continent, c.geosize_group, p.popsize_group
FROM countries_plus AS c
INNER JOIN pop_plus AS p
ON c.code = p.country_code;
```

LEFT JOIN, RIGHT JOIN

```
SELECT c1.name AS city, code, c2.name AS country, region, city_proper_pop
FROM cities AS c1
INNER JOIN countries AS c2
ON c1.country_code = c2.code
ORDER BY code DESC;


SELECT region, AVG(gdp_percapita) AS avg_gdp
FROM countries AS c
LEFT JOIN economies AS e
ON c.code = e.code
WHERE e.year = 2010
GROUP BY region
ORDER BY avg_gdp DESC;

-- 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 countries
ON languages.code = countries.code
RIGHT JOIN cities
ON countries.code = cities.country_code
ORDER BY city, language;
```

FULL JOIN

```
SELECT left_table.id, right_table.id, left_table.val, right_table.val
FROM left_table
FULL JOIN right_table
USING (id)
```

EX: FULL JOIN vs. LEFT JOIN vs. INNER JOIN

CASE WHEN left table has less rows than right table

FULL JOIN:

```
SELECT name AS country, code, region, basic_unit
FROM countries
FULL JOIN currencies
USING (code)
WHERE region = 'North America' OR region IS NULL
ORDER BY region;
```

QUERY RESULT | COUNTRIES | CURRENCIES

country	code	region	basic_unit
Canada	CAN	North America	Canadian dollar
United States	USA	North America	United States dollar
Bermuda	BMU	North America	Bermudian dollar
Greenland	GRL	North America	null

null	TMP	null	United States dollar

Showing 17 out of 17 rows

LEFT JOIN:

```
SELECT name AS country, code, region, basic_unit
FROM countries
LEFT JOIN currencies
USING (code)
WHERE region = 'North America' OR region IS NULL
ORDER BY region;
```

QUERY RESULT | COUNTRIES | CURRENCIES

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

Showing 4 out of 4 rows

INNER JOIN:

```
SELECT name AS country, code, region, basic_unit
FROM countries
INNER JOIN currencies
USING (code)
WHERE region = 'North America' OR region IS NULL
ORDER BY region;
```

QUERY RESULT | COUNTRIES | CURRENCIES

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

Showing 3 out of 3 rows

CASE WHEN left table has more rows than right table

FULL JOIN:

```
SELECT countries.name, code, languages.name AS language
FROM languages
FULL JOIN countries
USING (code)
WHERE countries.name LIKE 'V%' OR countries.name IS NULL
ORDER BY countries.name;
```

QUERY RESULT | LANGUAGES | COUNTRIES

name	code	language
Vanuatu	VUT	Tribal Languages
Vanuatu	VUT	English
Vanuatu	VUT	French
Vanuatu	VUT	Other
Vanuatu	VUT	Bislama

Showing 53 out of 53 rows

LEFT JOIN:

```
SELECT countries.name, code, languages.name AS language
FROM languages
LEFT JOIN countries
```

LEFT JOIN countries
USING (code)
WHERE countries.name LIKE 'V%' OR countries.name IS NULL
ORDER BY countries.name;

QUERY RESULT | LANGUAGES | COUNTRIES

name	code	language
Vanuatu	VUT	English
Vanuatu	VUT	Other
Vanuatu	VUT	French
Vanuatu	VUT	Tribal Languages
Vanuatu	VUT	Bislama

Showing 51 out of 51 rows

INNER JOIN:

SELECT countries.name, code, languages.name AS language
FROM languages
INNER JOIN countries
USING (code)
WHERE countries.name LIKE 'V%' OR countries.name IS NULL
ORDER BY countries.name;

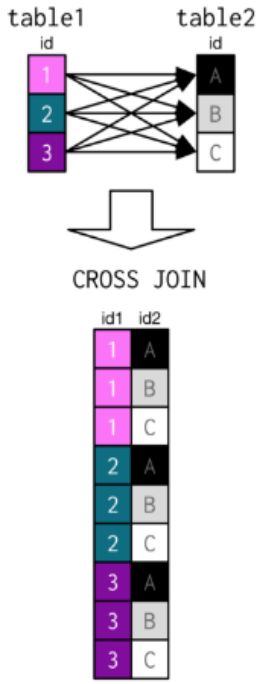
QUERY RESULT | LANGUAGES | COUNTRIES

name	code	language
Vanuatu	VUT	English
Vanuatu	VUT	Other
Vanuatu	VUT	French

Vanuatu	VUT	Tribal Languages
Vanuatu	VUT	Bislama

Showing 51 out of 51 rows

CROSS JOIN: produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used alng with CROSS JOIN.



SELECT table1, table2
FROM table1 AS t1
CROSS JOIN table2 AS t2
WHERE t1.var IN ('AA','BB')

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

```
SELECT c.name AS city, l.name AS language
FROM cities AS c
CROSS JOIN languages AS l
WHERE c.name LIKE 'Hyder%'
```

QUERY RESULT | LANGUAGES | CITIES

city	language
Hyderabad (India)	Dari
Hyderabad	Dari
Hyderabad (India)	Pashto
Hyderabad	Pashto
Hyderabad (India)	Turkic

Showing 100 out of 1910 rows

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 c.name AS city, l.name AS language
FROM cities AS c
INNER JOIN languages AS l
on c.country_code = l.code
WHERE c.name LIKE 'Hyder%';
```

QUERY RESULT | LANGUAGES | CITIES

city	language
Hyderabad (India)	Hindi
Hyderabad (India)	Bengali
Hyderabad (India)	Telugu
Hyderabad (India)	Marathi

Hyderabad (India)	Tamil

Showing 25 out of 25 rows

Practice:

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

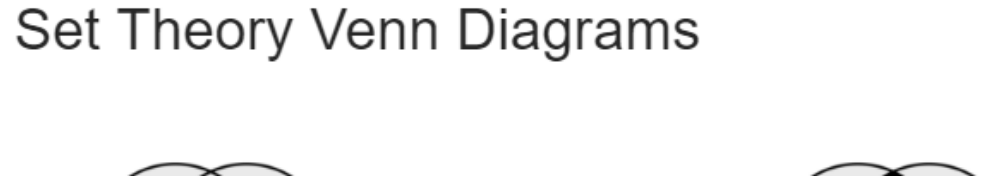
```
SELECT c.name AS country, region, life_expectancy AS life_exp
FROM countries AS c
LEFT JOIN populations AS p
ON c.code = p.country_code
WHERE p.year = 2010
ORDER BY life_exp
LIMIT 5;
```

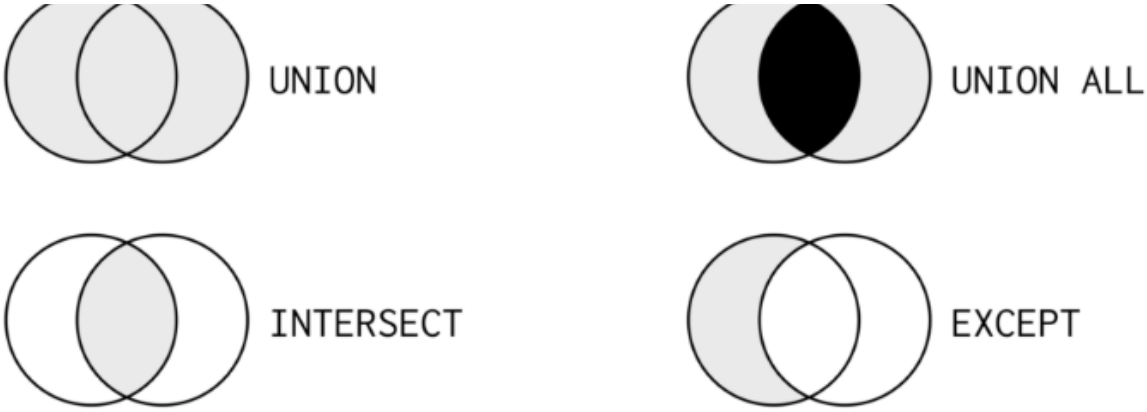
QUERY RESULT | LANGUAGES | CURRENCIES | POPULATIONS | ECON

country	region	life_exp
Lesotho	Southern Africa	47.4834
Central African Republic	Central Africa	47.6253
Sierra Leone	Western Africa	48.229
Swaziland	Southern Africa	48.3458
Zimbabwe	Eastern Africa	49.5747

Showing 5 out of 5 rows

STATE OF UNION





Note: Fields included in the operation must be the same type. You can't put number in a character

UNION selects only distinct values by default. To allow duplicate values, use UNION ALL

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

Ex:
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;
```

fields followed by ORDER BY should be the same as the first table

INTERSECT:

One field:

```
SELECT id
FROM left_one
INTERSECT
SELECT id
FROM right_one;
```

Two fields: INTERSECT looks both fields in the search, it only looks records in common not keywords

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

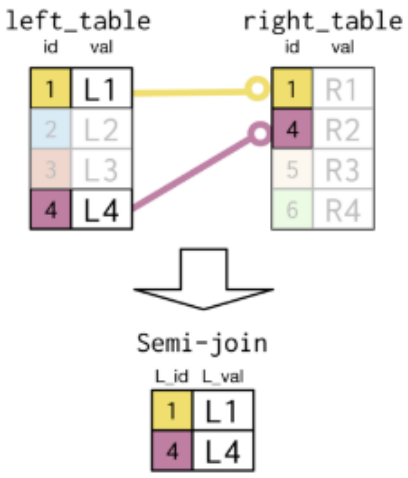
EXCEPTional: Shows only records from left table that is not in the right table

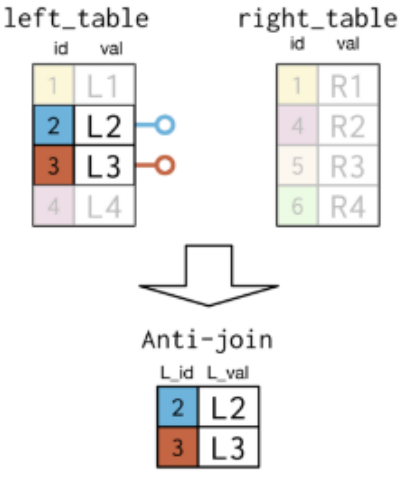
Ex:
Get the names of cities in `cities` which are not noted as capital cities in `countries` as a single field result.

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

Semi-jons: Similar to WHERE clause. (Subsidiary)

Semi-join and anti-join diagrams





Ex:

```
SELECT president, country, continent
FROM presidents
WHERE country IN
      (SELECT name
      FROM states
      WHERE indep_year < 1800);
```

Ex2:

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

1) 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';
```

2) select only unique languages by name appearing in the `languages` table.

```
SELECT DISTINCT name
FROM languages
ORDER BY name;
```

3) Now combine the previous two queries into one query using `WHERE` to determine the unique 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 the problems can be solved with 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 has the same result as above.

Anti-join: It is useful to identifying which records are causing an incorrect number of records to appear in join queries.

Ex: Your goal is to identify the currencies used in Oceanian countries!

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

```
SELECT COUNT(DISTINCT name)
FROM countries
WHERE continent = 'Oceania';
```

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

```
SELECT c1.code, c1.name, basic_unit AS currency
FROM countries AS c1
INNER JOIN currencies AS c2
ON c1.code = c2.code
WHERE continent = 'Oceania';
```

QUERY RESULT | COUNTRIES | CURRENCIES

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

Showing 15 out of 15 rows

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

```
SELECT code, name
```

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

All-include Exercise:

- 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
SELECT name
-- alias the table where city name resides
FROM cities AS c1
-- choose only records matching the result of multiple set theory clauses
WHERE 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
);
```

Subquerys inside WHERE and SELECT clauses

Subquerys inside WHERE clauses

Ex:

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

Ex:

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

```
-- select the appropriate fields
SELECT name, country_code, urbanarea_pop
-- from the cities table
FROM cities
-- with city name in the field of capital cities
WHERE name IN
  (SELECT capital
   FROM countries)
ORDER BY urbanarea_pop DESC;
```

QUERY RESULT | CITIES | COUNTRIES

name	country_code	urbanarea_pop
Beijing	CHN	21516000
Dhaka	BGD	14543100
Tokyo	JPN	13513700
Moscow	RUS	12197600
Cairo	EGY	10230400

Showing 66 out of 66 rows

Subquery inside SELECT clauses:

Ex:

```
/*
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,
  (SELECT COUNT(*)
   FROM cities
   WHERE countries.code = cities.country_code) AS cities_num
FROM countries
ORDER BY cities_num DESC, country
LIMIT 9;
```

QUERY RESULT | CITIES | COUNTRIES

country	cities_num
China	36
India	18
Japan	11
Brazil	10
Pakistan	9

Showing 9 out of 9 rows

Subquery inside FROM clauses:

Ex:

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.)

Step 1:
Begin by determining for each country code how many languages are listed in the `languages` table using `SELECT`, `FROM`, and `GROUP BY`.

```
SELECT code, COUNT(*) AS lang_num
FROM languages
GROUP BY code;
```

Step 2:
Include the previous query (aliased as `subquery`) as a subquery in the `FROM` clause of a new query.

```
SELECT local_name, 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;
```

Advanced Subquery:
Ex:

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    |
```

<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.

Step1:

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;
```

Step2:

- 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_infgrouped by continent.

```
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;
```

48.684
9.784
39.403
21.858
7.524
121.738

Showing 6 out of 6 rows

Step3:

- 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`.

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

QUERY RESULT | COUNTRIES | ECONOMIES

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

Showing 6 out of 6 rows

Subquery Challenge:

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.

```
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;
```

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

Showing 26 out of 26 rows

Final Challenge 1:
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.

- 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, total_investment, imports
FROM countries AS c
LEFT JOIN economies AS e
ON c.code = e.code AND c.code IN (SELECT code
                                  FROM languages
                                  WHERE official = 'true')
WHERE year = 2015 AND region = 'Central America'
ORDER BY c.name;
```

name	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

Showing 7 out of 7 rows

Final Challenge 2:
Let's ease up a bit and calculate the average fertility rate for each region in 2015.

- 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
LEFT JOIN populations AS p
ON c.code = p.country_code
WHERE p.year = 2015
GROUP BY region, continent
ORDER BY avg_fert_rate;
```

QUERY RESULT | LANGUAGES | POPULATIONS | CURRENCIES | ECONOM

region	continent	avg_fert_rate
Southern Europe	Europe	1.42610000371933
Eastern Europe	Europe	1.49088890022702
Baltic Countries	Europe	1.60333331425985
Eastern Asia	Asia	1.62071430683136

Western Europe	Europe	1.6325000077486

Showing 23 out of 23 rows

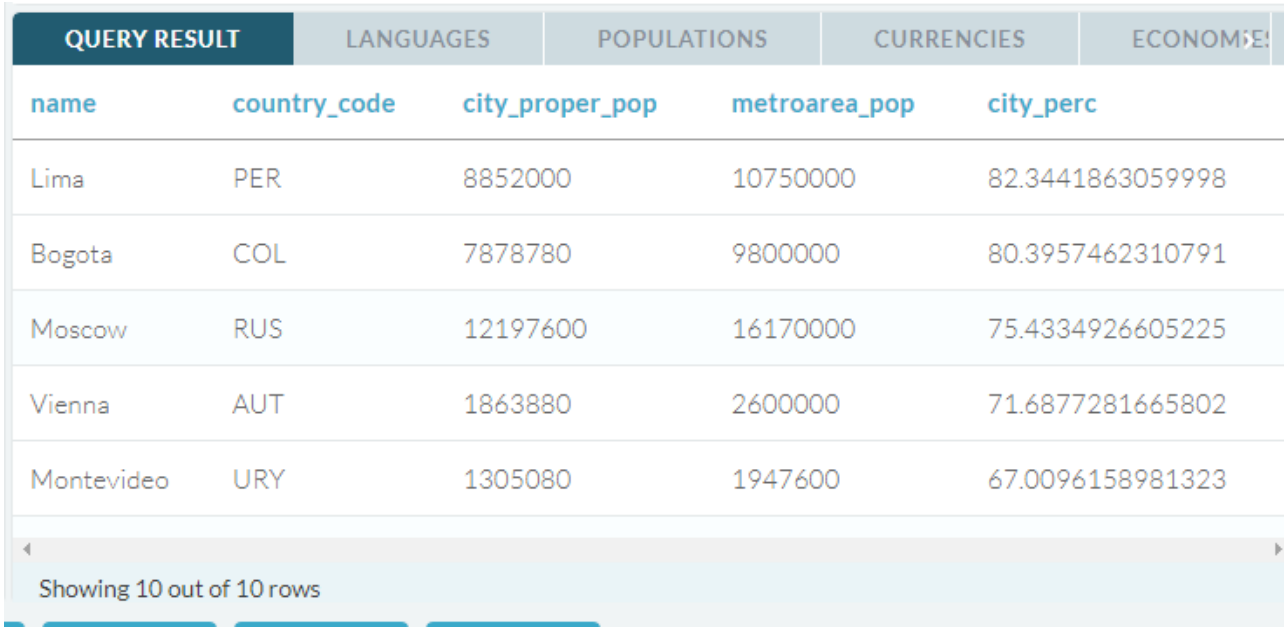
Final Challenge 3:

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`.

- 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 * 100.0 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;
```



QUERY RESULT | LANGUAGES | POPULATIONS | CURRENCIES | ECONOMIES

name	country_code	city_proper_pop	metroarea_pop	city_perc
Lima	PER	8852000	10750000	82.3441863059998
Bogota	COL	7878780	9800000	80.3957462310791
Moscow	RUS	12197600	16170000	75.4334926605225
Vienna	AUT	1863880	2600000	71.6877281665802
Montevideo	URY	1305080	1947600	67.0096158981323

Showing 10 out of 10 rows