SQL Bootcamp for Data Science

Mona Khalil @ AI+

September 30, 2020

Session Overview

1. Relational Databases and Foundational SQL
2. Combining Data From Multiple Tables and Columns
3. Layering your Transformation with Subqueries
4. Transforming your Data for Analysis

Complete the following

- Download the databases from Google Drive
  - **world_governance_indicators.db**
  - **global_powerplants.db**
- Download and install **SQLite Studio** sqlitetutorial.net/download-install-sqlite/

About

- Data Scientist at Greenhouse Software
  - Teaching SQL for >4 years
  - ***Intermediate SQL*** instructor with DataCamp
  - Passionate about analytics education, data science ethics, and effective use of research methods

Twitter: mona_kay_

1

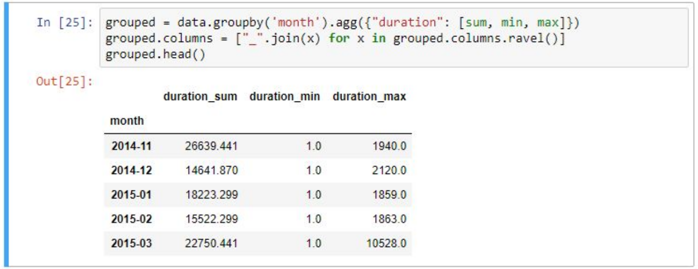
Foundations

- Relational databases
- Basic SQL syntax
- Aggregating, filtering, and sorting data

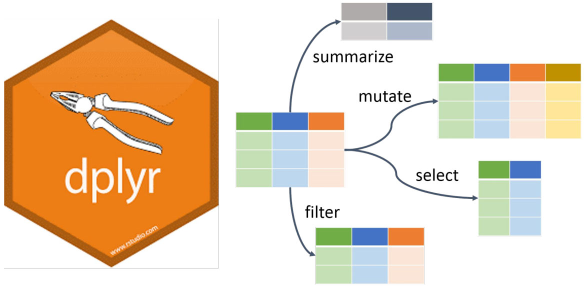
Why SQL?

- **Structured Query Language** (SQL) has been used to retrieve and shape information from relational databases for 40+ years
  - Most commonly requested/utilized skill among data scientists and data engineers
  - Foundation of data manipulation and analysis in Python, R, Tableau, etc.
    - Pandas in Python and dplyr in R use similar syntax
    - Many proprietary query languages are based on SQL

room_name_full FROM tbl_room WHERE room_checked='0'"
$uncheckedRoom["room_name_full"]."' class
room_name_full"];
on tbl_disposition.disposition_user = u.user_ID
MacBook

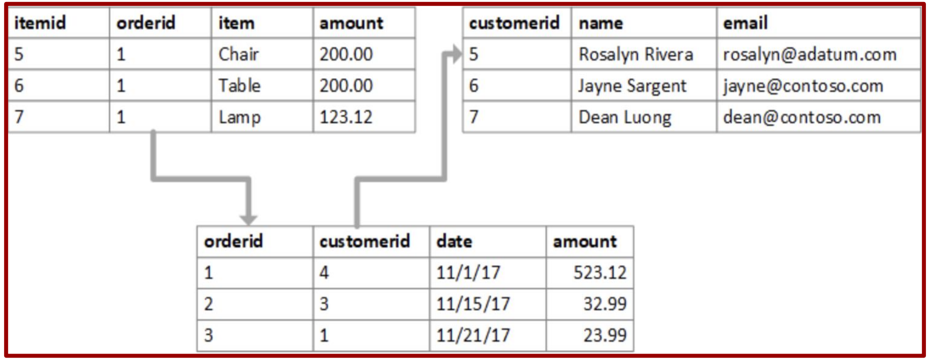


```
In [25]: grouped = data.groupby('month').agg({"duration": [sum, min, max]})
         grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]
         grouped.head()

Out[25]:
                duration_sum  duration_min  duration_max
         month
         2014-11   26639.441           1.0        1940.0
         2014-12   14641.870           1.0        2120.0
         2015-01   18223.299           1.0        1859.0
         2015-02   15522.299           1.0        1863.0
         2015-03   22750.441           1.0       10528.0
```



Relational Databases

- A **relational database** is a set of tables with relationships that determine how the information in each *relates* to other tables
  - Tables are connected by **keys** (columns that identify data across tables)
    - **Primary key** - unique record identifier
    - **Foreign key** - record identifier for another table
  - Stores data for efficient and broad accessibility and addition of new records



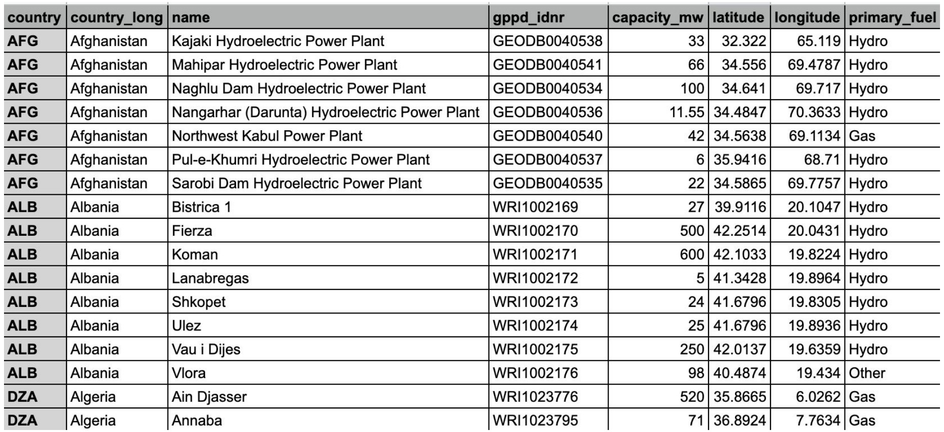
itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

Relational Databases

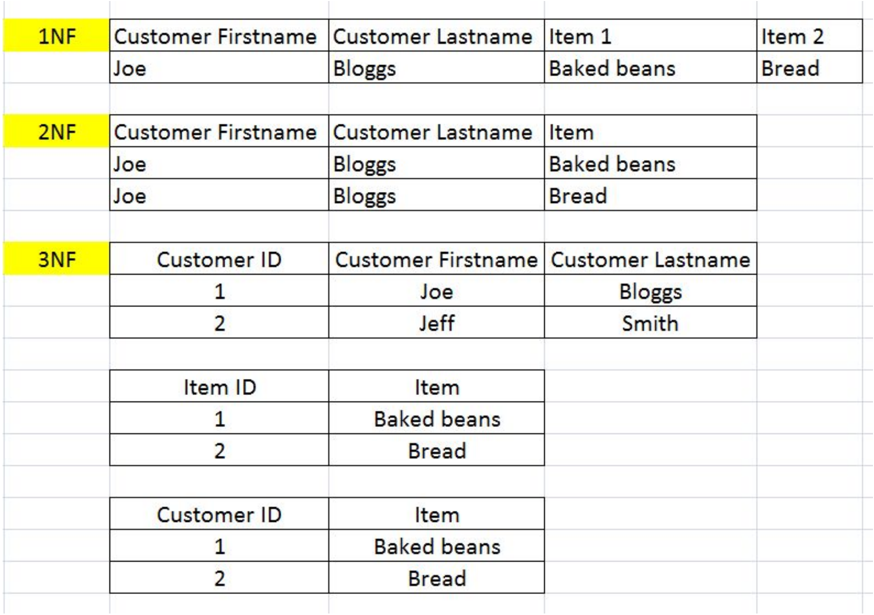
- Data storage as flat files is limited in usability
    - Storage is on the level of the most granular piece of data
    - Repeat/duplicate information
        - Larger file size
    - Less than optimal for most questions you'll have from the data
        - When are records updated?
        - What information was updated?



country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel
AFG	Afghanistan	Kajaki Hydroelectric Power Plant	GEODB0040538	33	32.322	65.119	Hydro
AFG	Afghanistan	Mahipar Hydroelectric Power Plant	GEODB0040541	66	34.556	69.4787	Hydro
AFG	Afghanistan	Naghlu Dam Hydroelectric Power Plant	GEODB0040534	100	34.641	69.717	Hydro
AFG	Afghanistan	Nangarhar (Darunta) Hydroelectric Power Plant	GEODB0040536	11.55	34.4847	70.3633	Hydro
AFG	Afghanistan	Northwest Kabul Power Plant	GEODB0040540	42	34.5638	69.1134	Gas
AFG	Afghanistan	Pul-e-Khumri Hydroelectric Power Plant	GEODB0040537	6	35.9416	68.71	Hydro
AFG	Afghanistan	Sarobi Dam Hydroelectric Power Plant	GEODB0040535	22	34.5865	69.7757	Hydro
ALB	Albania	Bistrica 1	WRI1002169	27	39.9116	20.1047	Hydro
ALB	Albania	Fierza	WRI1002170	500	42.2514	20.0431	Hydro
ALB	Albania	Koman	WRI1002171	600	42.1033	19.8224	Hydro
ALB	Albania	Lanabregas	WRI1002172	5	41.3428	19.8964	Hydro
ALB	Albania	Shkopet	WRI1002173	24	41.6796	19.8305	Hydro
ALB	Albania	Ulez	WRI1002174	25	41.6796	19.8936	Hydro
ALB	Albania	Vau i Dijes	WRI1002175	250	42.0137	19.6359	Hydro
ALB	Albania	Vlora	WRI1002176	98	40.4874	19.434	Other
DZA	Algeria	Ain Djasser	WRI1023776	520	35.8665	6.0262	Gas
DZA	Algeria	Annaba	WRI1023795	71	36.8924	7.7634	Gas

Relational Databases

- Data is sorted into relational databases in a process called normalization
  - Limit tables to one topic/purpose
  - Store **records** (pieces of information) as **rows**
  - Remove duplicates and optimize for space
  - Store data for widest accessibility and ease of adding new records



1NF	Customer Firstname	Customer Lastname	Item 1	Item 2
	Joe	Bloggs	Baked beans	Bread

2NF	Customer Firstname	Customer Lastname	Item
	Joe	Bloggs	Baked beans
	Joe	Bloggs	Bread

3NF	Customer ID	Customer Firstname	Customer Lastname
	1	Joe	Bloggs
	2	Jeff	Smith

Item ID	Item
1	Baked beans
2	Bread

Customer ID	Item
1	Baked beans
2	Bread

Common SQL Databases

- Several common SQL databases
  - PostgreSQL
  - Microsoft SQL Server
  - Oracle
  - MySQL
- Most cloud data warehouses use one of these types of databases
- Store data in **relational databases**
  - Minor differences in SQL syntax used to retrieve and transform data
  - Minor differences in data types

SQLite

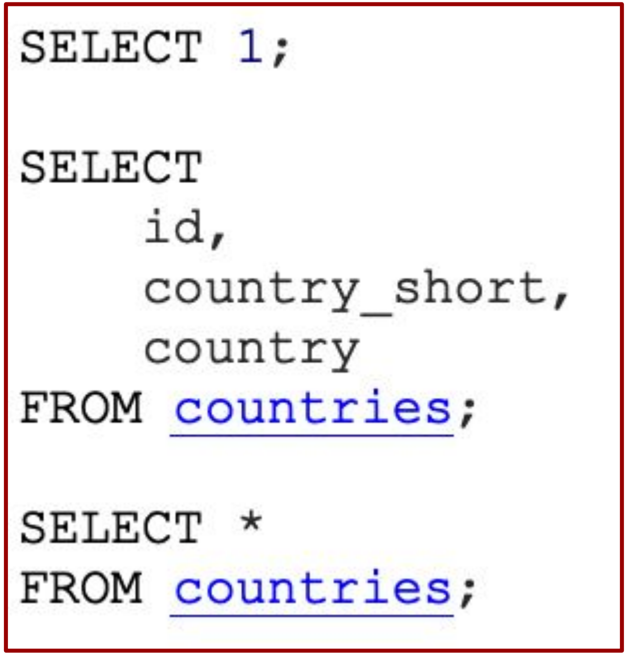
- SQLite is an open-source standalone database management system
  - Easy tool to transition from using flat files (CSVs)
  - Local setup with easy file connection and creation

SQLite Command Line Interface (CLI): sqlitetutorial.net/download-install-sqlite/

SQLite Studio: github.com/pawelsalawa/sqlitestudio/releases

SQL Syntax

- SQL is designed to **SELECT** information **FROM** a source (e.g., a table)
  - Can be used as a calculator (first example)
  - Select ***specific columns*** from a table
  - Select ***all columns*** from a table (**SELECT ***)

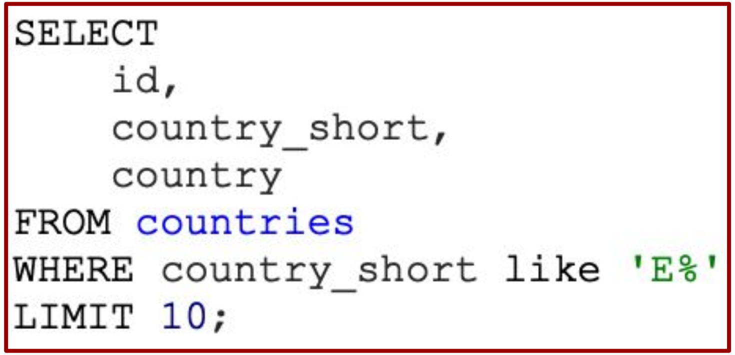


```
SELECT 1;

SELECT
    id,
    country_short,
    country
FROM countries;

SELECT *
FROM countries;
```

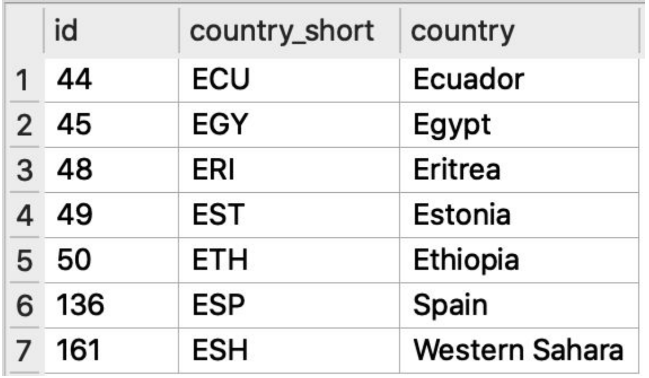
SQL Syntax

SQL allows you to filter, sort, combine, and manipulate information to retrieve what you need.

- Filter using the `WHERE` clause
  - Filter with mathematical operators
    `WHERE value >= 10`
  - Filter based on text values or patterns
    `WHERE name = 'USA'`
    `WHERE name like '%e%'`
  - Filter based on a list
    `WHERE value IN (7, 14, 21, 28)`
- ****Tip:** use `LIMIT 10;` to limit the number of records you return



```
SELECT
     id,
     country_short,
     country
FROM countries
WHERE country_short like 'E%'
LIMIT 10;
```



	id	country_short	country
1	44	ECU	Ecuador
2	45	EGY	Egypt
3	48	ERI	Eritrea
4	49	EST	Estonia
5	50	ETH	Ethiopia
6	136	ESP	Spain
7	161	ESH	Western Sahara

Exercise 1 (3 minutes)

You have access to two databases:

- `global_powerplants.db` contains data on characteristics and output of thousands of power plants around the world. We'll use this for all exercises throughout the course.
- `world_governance_indicators.db` is a ***time series database*** about governance policies across world nations from 1996 to 2018. You can use this to practice on your own.

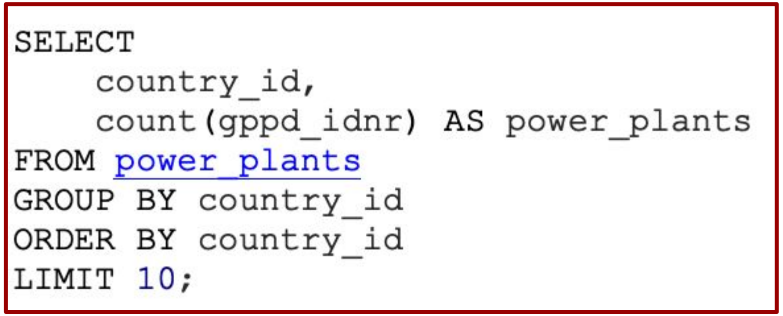
1. Select the first few records from the `power_generation` and `power_plants` tables.
   - Write down the columns you believe are shared between these tables.
2. Explore the characteristics of the `world_governance_indicators` database by querying the `sqlite_master` schema.

Aggregations

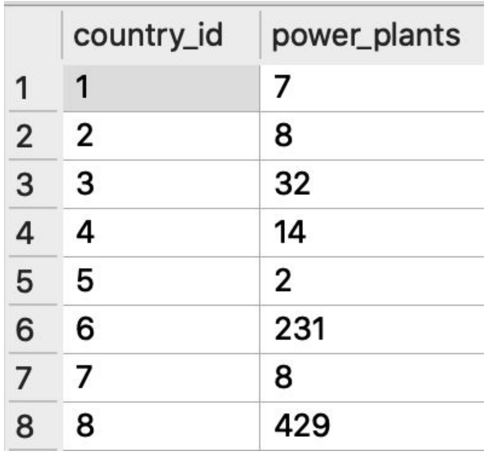
- Calculate aggregate/summary information (`COUNT`, `AVG`, `MIN`, `MAX`, etc.)
    - Count the number of records in a table

      ```
      SELECT count(*)
      FROM countries;
      ```

    - Find the number of records per group
- `GROUP BY` *all* columns not being aggregated
- **Tip:* alias an aggregate column using `AS`



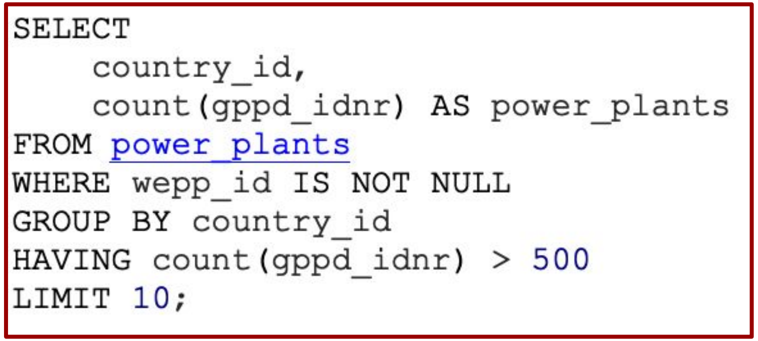
```
SELECT
    country_id,
    count(gppd_idnr) AS power_plants
FROM power_plants
GROUP BY country_id
ORDER BY country_id
LIMIT 10;
```



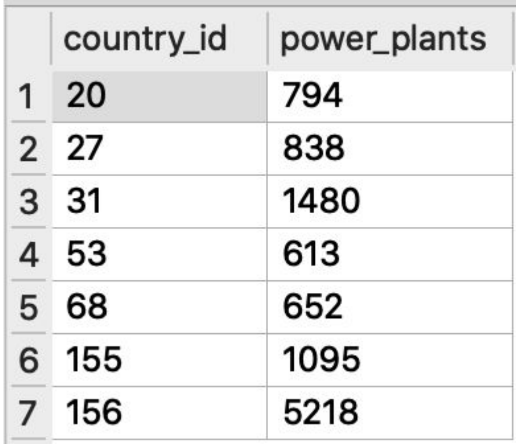
	country_id	power_plants
1	1	7
2	2	8
3	3	32
4	4	14
5	5	2
6	6	231
7	7	8
8	8	429

Filter Results

- Filter results *before* or *after* an aggregation
  - `WHERE` filters the raw table contents before the `GROUP BY`
  - `HAVING` filters results based on the values in the aggregate clause
- ***Tip*:** Use `SELECT DISTINCT` or `count(DISTINCT column)` to get unique values



```
SELECT
    country_id,
    count(gppd_idnr) AS power_plants
FROM power_plants
WHERE wepp_id IS NOT NULL
GROUP BY country_id
HAVING count(gppd_idnr) > 500
LIMIT 10;
```



	country_id	power_plants
1	20	794
2	27	838
3	31	1480
4	53	613
5	68	652
6	155	1095
7	156	5218

Exercise 2 (4 minutes)

1. How many unique types of `primary_fuel` exist in the `power_plants` table?
2. On average, which type of `primary_fuel` has the greatest `capacity_mw`?
3. Which `country_id` has the highest total `generation_gwh` in the `power_generation` table?

2

Combining Data

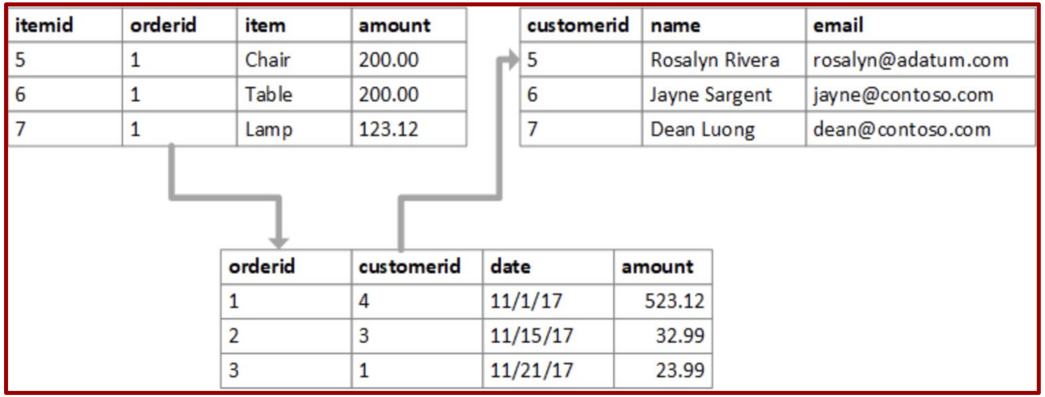
- Identifying keys in tables
- Choosing the correct join
- Performing mathematical calculations on data

Combining Data

The greatest value SQL provides is the ability to join information across multiple tables, schemas, and sources for a more comprehensive analysis.

Joins

- Combine information from multiple tables
  - Columns with shared information can be used as keys to join data across multiple tables
  - Multiple types of joins that determine shape, null values, and aggregate results in your final dataset



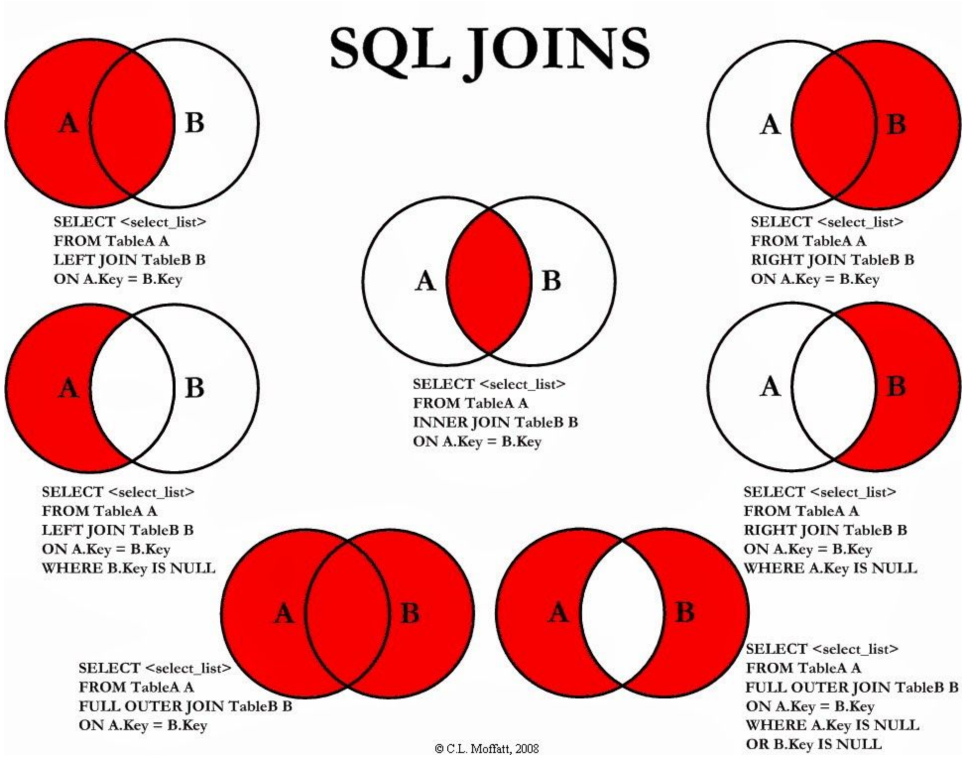
itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

Joins

- Four main types of joins
  - LEFT, RIGHT, OUTER, INNER
- ***Tip:*** Most commonly used joins are LEFT and INNER



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

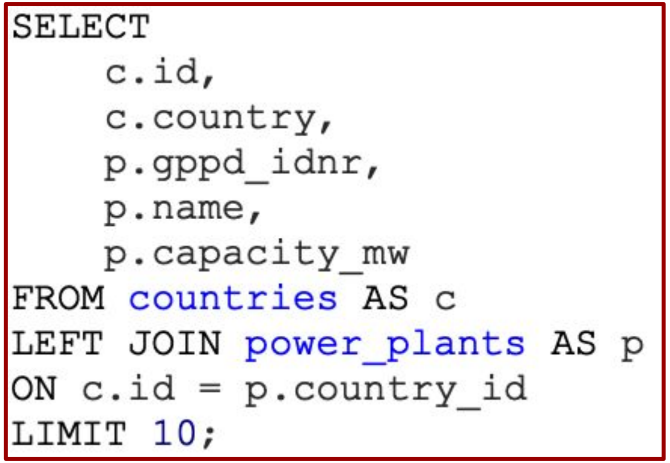
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

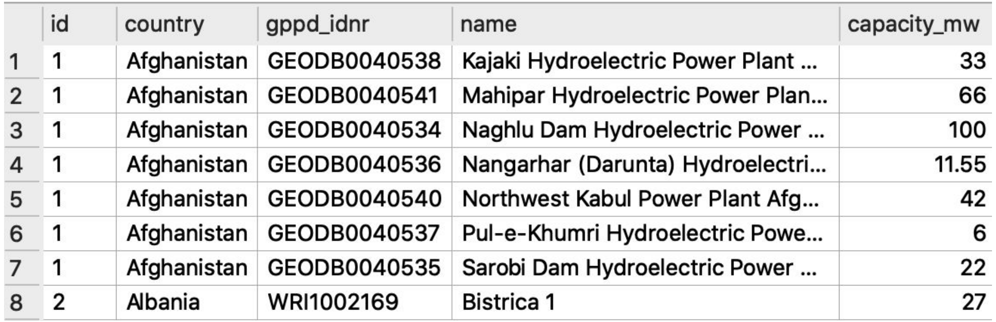
© C.L. Moffatt, 2008

Left Joins

- Retrieve ***all*** records in the left table and any in the right table with a matching key value
    - Name your “left” table first
    - Give each table an alias
    - Prefix each column with the table alias



```
SELECT
    c.id,
    c.country,
    p.gppd_idnr,
    p.name,
    p.capacity_mw
FROM countries AS c
LEFT JOIN power_plants AS p
ON c.id = p.country_id
LIMIT 10;
```



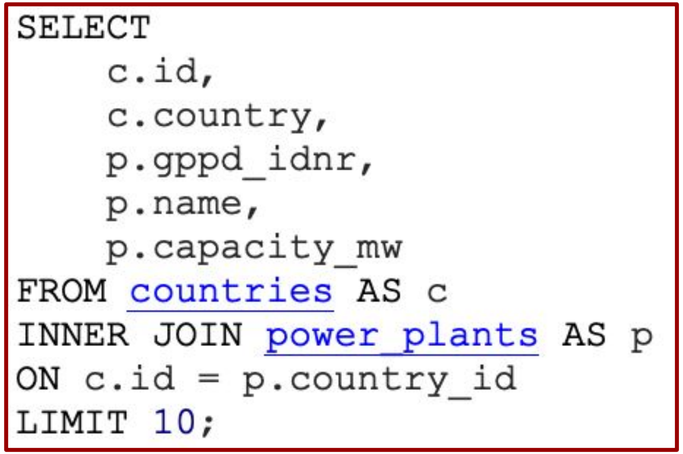
	id	country	gppd_idnr	name	capacity_mw
1	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33
2	1	Afghanistan	GEODB0040541	Mahipar Hydroelectric Power Plan...	66
3	1	Afghanistan	GEODB0040534	Naghlu Dam Hydroelectric Power ...	100
4	1	Afghanistan	GEODB0040536	Nangarhar (Darunta) Hydroelectri...	11.55
5	1	Afghanistan	GEODB0040540	Northwest Kabul Power Plant Afg...	42
6	1	Afghanistan	GEODB0040537	Pul-e-Khumri Hydroelectric Powe...	6
7	1	Afghanistan	GEODB0040535	Sarobi Dam Hydroelectric Power ...	22
8	2	Albania	WRI1002169	Bistrica 1	27

Exercise 3 (4 minutes)

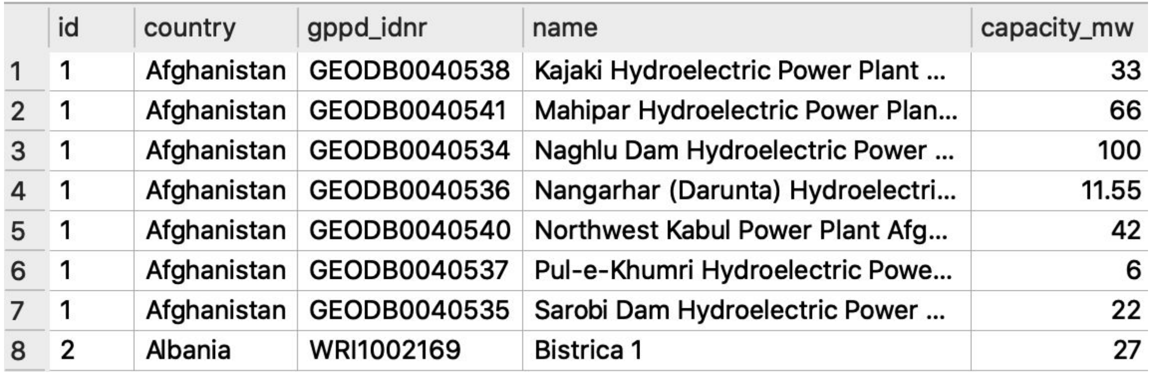
1. What type of join do you need if you want the number of power plants per country, including countries with no records in the `power_plants` table?
2. Select all information from the `countries` and `power_plants` tables using a left join. Limit your results to 100.

Inner Joins

- Retrieve only records that share a key value in *both* tables
  - Order of tables listed does not impact the results
  - Any record without a value in either table is excluded



```
SELECT
    c.id,
    c.country,
    p.gppd_idnr,
    p.name,
    p.capacity_mw
FROM countries AS c
INNER JOIN power_plants AS p
ON c.id = p.country_id
LIMIT 10;
```



	id	country	gppd_idnr	name	capacity_mw
1	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33
2	1	Afghanistan	GEODB0040541	Mahipar Hydroelectric Power Plan...	66
3	1	Afghanistan	GEODB0040534	Naghlu Dam Hydroelectric Power ...	100
4	1	Afghanistan	GEODB0040536	Nangarhar (Darunta) Hydroelectri...	11.55
5	1	Afghanistan	GEODB0040540	Northwest Kabul Power Plant Afg...	42
6	1	Afghanistan	GEODB0040537	Pul-e-Khumri Hydroelectric Powe...	6
7	1	Afghanistan	GEODB0040535	Sarobi Dam Hydroelectric Power ...	22
8	2	Albania	WRI1002169	Bistrica 1	27

Exercise 4 (5 minutes)

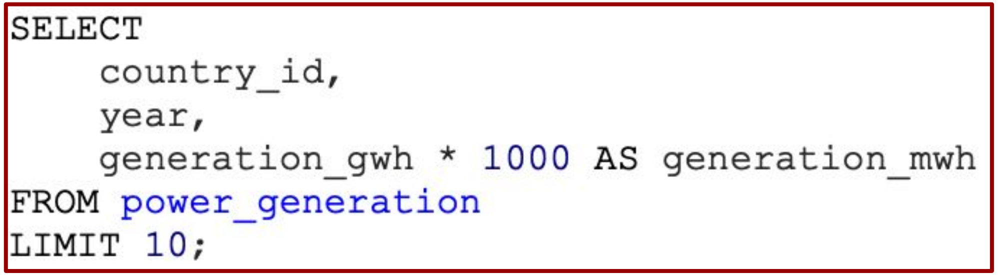
1. How many power plants are in each country? Join the `power_plants` table to the `countries` table to get the country name.

2. What are the top 3 countries with the most power plants in the database? You can use `DESC` to sort your results in descending order.

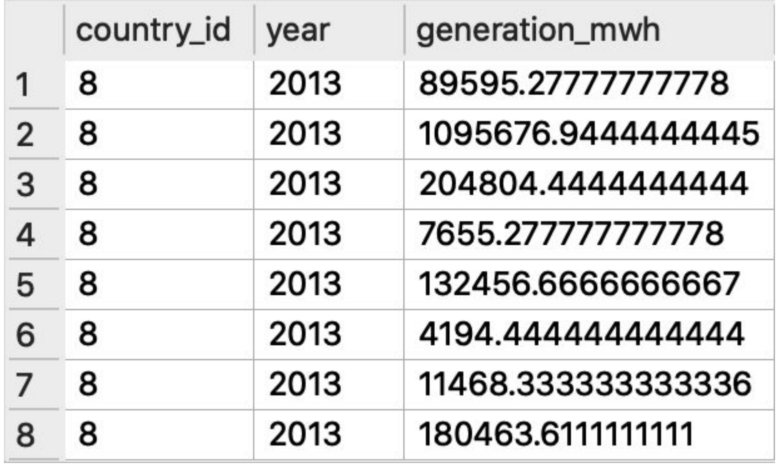
3. Which country has the most unique sources in the `data_sources` table?

Combining Information in Columns

- You can perform mathematical calculations on 1 or more columns without aggregations
  - Convert units
  - Calculate proportions/averages between columns
  - Manually calculate averages/ sums across multiple columns



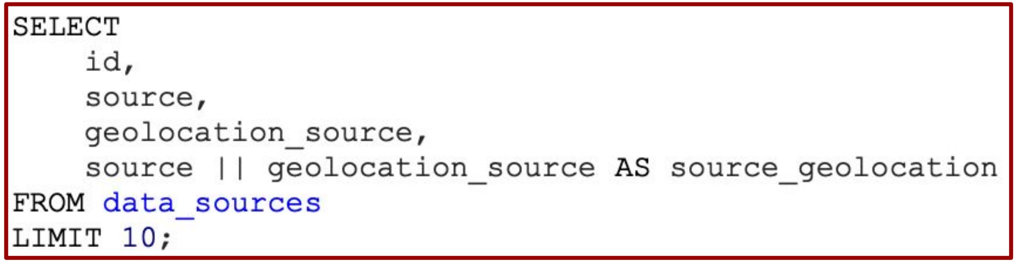
```
SELECT
    country_id,
    year,
    generation_gwh * 1000 AS generation_mwh
FROM power_generation
LIMIT 10;
```



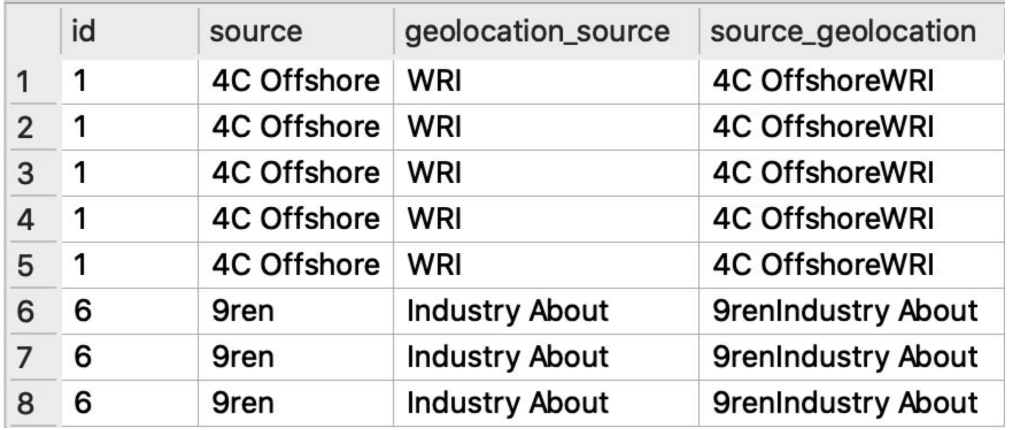
	country_id	year	generation_mwh
1	8	2013	89595.27777777778
2	8	2013	1095676.9444444445
3	8	2013	204804.4444444444
4	8	2013	7655.277777777778
5	8	2013	132456.66666666667
6	8	2013	4194.444444444444
7	8	2013	11468.333333333336
8	8	2013	180463.6111111111

Combining Information in Columns

- Manipulate text in columns
  - Concatenate multiple columns
  - Generate new columns from pieces of data
  - Perform regular expression searches for text patterns
- **_*Tip:_** Different versions of SQL (i.e., PostgreSQL vs. SQL Server) have different text manipulation functions, but most versions offer the same types of actions



```
SELECT
    id,
    source,
    geolocation_source,
    source || geolocation_source AS source_geolocation
FROM data_sources
LIMIT 10;
```



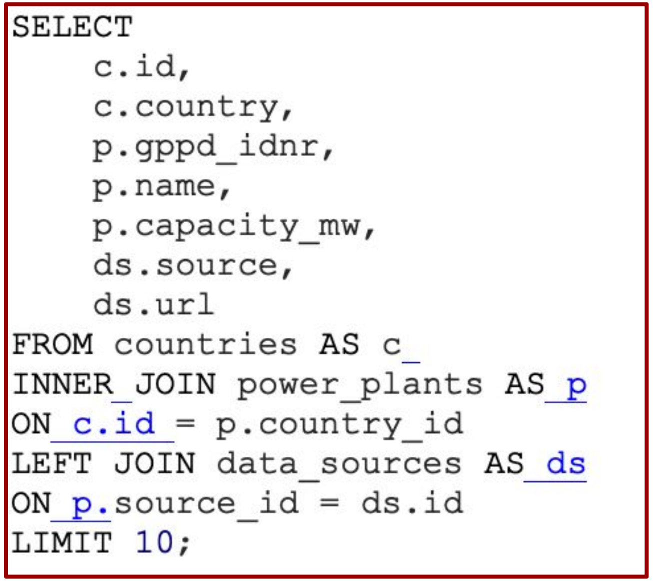
	id	source	geolocation_source	source_geolocation
1	1	4C Offshore	WRI	4C OffshoreWRI
2	1	4C Offshore	WRI	4C OffshoreWRI
3	1	4C Offshore	WRI	4C OffshoreWRI
4	1	4C Offshore	WRI	4C OffshoreWRI
5	1	4C Offshore	WRI	4C OffshoreWRI
6	6	9ren	Industry About	9renIndustry About
7	6	9ren	Industry About	9renIndustry About
8	6	9ren	Industry About	9renIndustry About

Exercise 5 (5 minutes)

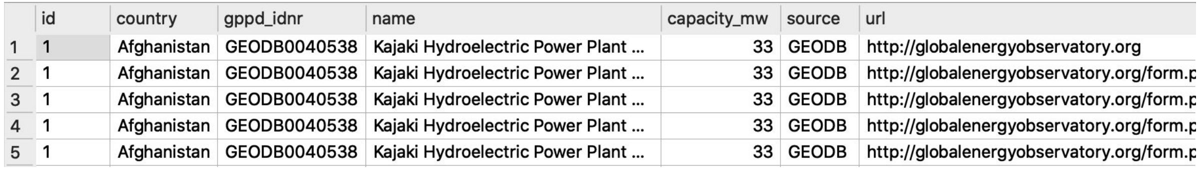
1. What is the oldest power plant in the `power_plants` table? Use the `commissioning_year` column to find out.
2. Create a new column called `fuels` that combines the `primary_fuel` and `other_fuel`.
3. Trim down the `url` column in the `data_sources` table using the `REPLACE()` function to remove `http://`. You can read about how it works here: https://www.sqlitetutorial.net/sqlite-replace-function/

Wrapping Up Joins

- You can join together as many tables as you want in multiple join combinations
  - Decide which keys you need to join to which table. Insufficient keys may lead to duplicates or errors!
  - The more joins you add, the slower your query will run.



```
SELECT
    c.id,
    c.country,
    p.gppd_idnr,
    p.name,
    p.capacity_mw,
    ds.source,
    ds.url
FROM countries AS c
INNER JOIN power_plants AS p
ON c.id = p.country_id
LEFT JOIN data_sources AS ds
ON p.source_id = ds.id
LIMIT 10;
```



	id	country	gppd_idnr	name	capacity_mw	source	url
1	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33	GEODB	http://globalenergyobservatory.org
2	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33	GEODB	http://globalenergyobservatory.org/form.p
3	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33	GEODB	http://globalenergyobservatory.org/form.p
4	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33	GEODB	http://globalenergyobservatory.org/form.p
5	1	Afghanistan	GEODB0040538	Kajaki Hydroelectric Power Plant ...	33	GEODB	http://globalenergyobservatory.org/form.p

15 Minute Break

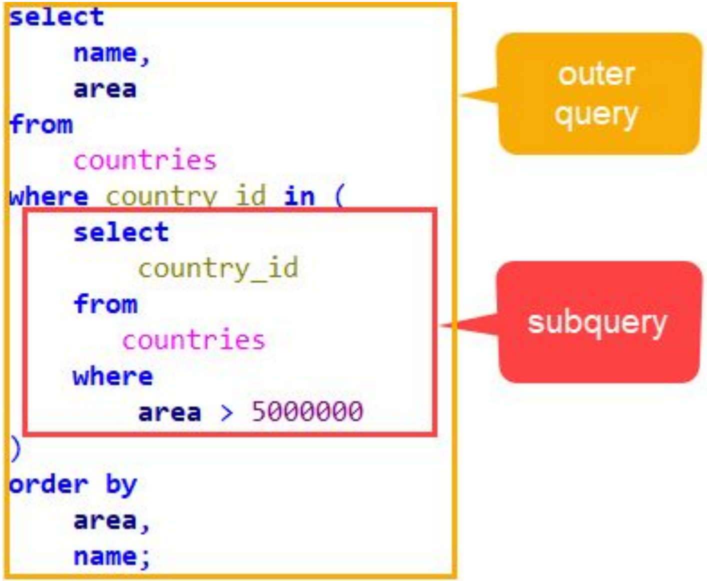
3

Layering Transformations

- Subqueries
- Common Table Expressions (CTEs)
- Window Functions

Subqueries

- Versatile way of transforming data in multiple ways/steps
  - Standardizing your data
  - Filtering based on a list
- Machine Learning example
  - An excellent feature for a predictive model is the previous average of your output value
  - Generate a 30-day rolling average prior to the date in your new prediction set



```
select
    name,
    area
from
    countries
where country_id in (
    select
        country_id
    from
        countries
    where
        area > 5000000
)
order by
    area,
    name;
```

Types of Subqueries

- **`FROM`**
  - Transform your data (filter, aggregate, etc)
  - Nest it in your **`FROM`** statement between parentheses
  - Select from this nested query like any other table in your database

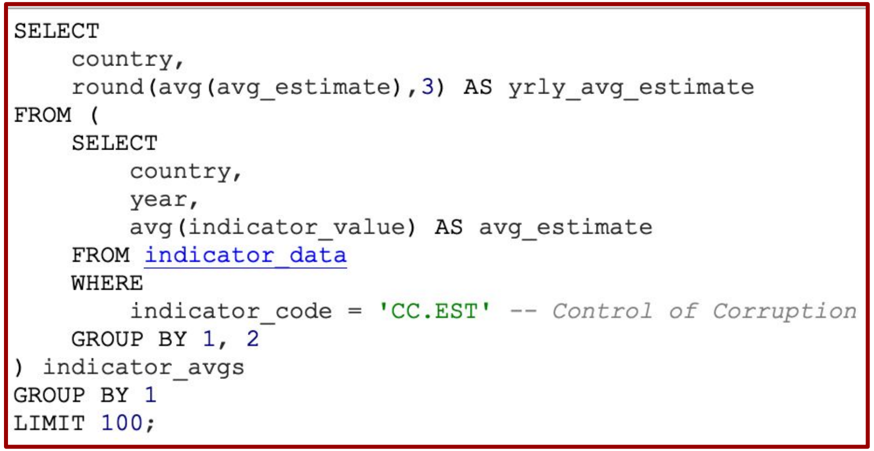
- **`WHERE/HAVING`**
  - Generate a list of values (e.g., customer IDs) to include or exclude in your final dataset

- **`SELECT`**
  - Calculate an aggregate value *without* aggregating the entire dataset*

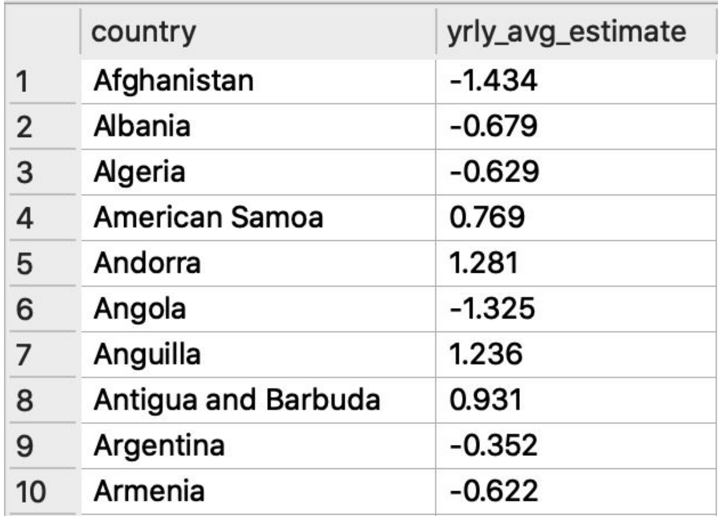
*Can also be accomplished with a window function

Subquery in FROM

- Write inner query performing first step of transformation/aggregation
- Place inner query inside parentheses in the `FROM` statement, and give it an alias
- Retrieve/transform information from the subquery in the main query
- **Tip:*** Use two dashes `--` to write single-line comments in your query
- ****Tip:*** You can group and order by column ***numbers*** or column ***names***.



```
SELECT
    country,
    round(avg(avg_estimate),3) AS yrly_avg_estimate
FROM (
    SELECT
        country,
        year,
        avg(indicator_value) AS avg_estimate
    FROM indicator_data
    WHERE
        indicator_code = 'CC.EST' -- Control of Corruption
    GROUP BY 1, 2
) indicator_avgs
GROUP BY 1
LIMIT 100;
```



	country	yrly_avg_estimate
1	Afghanistan	-1.434
2	Albania	-0.679
3	Algeria	-0.629
4	American Samoa	0.769
5	Andorra	1.281
6	Angola	-1.325
7	Anguilla	1.236
8	Antigua and Barbuda	0.931
9	Argentina	-0.352
10	Armenia	-0.622

Exercise 6 (5 minutes)

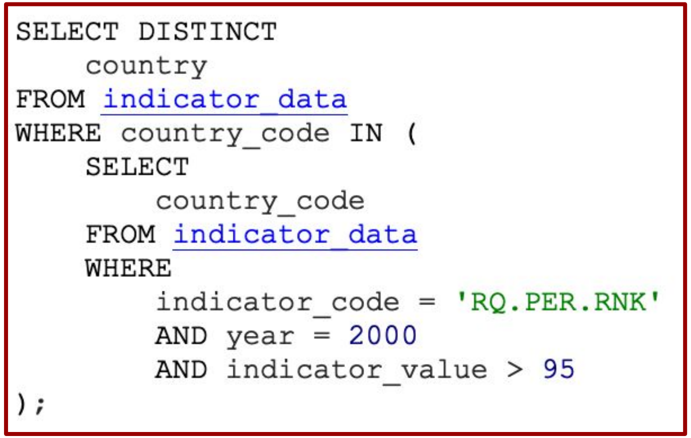
1. Calculate the ***standardized average*** power generation across each year. Complete this by first calculating the average in each country per year using a subquery. *Save this query for a later exercise!

2. What year in the United States has the highest total power generation?

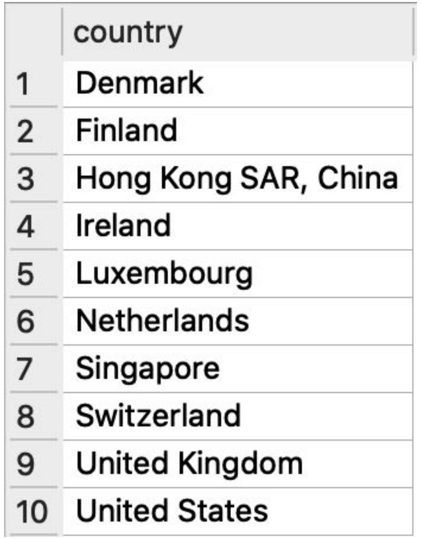
Subquery in WHERE

Subqueries in **`WHERE`** are used to create dynamic filtering lists

- Avoid hard-coding values into queries to accurately capture changes
- Generate a list of IDs to choose based on complex filtering conditions



```
SELECT DISTINCT
    country
FROM indicator_data
WHERE country_code IN (
    SELECT
        country_code
    FROM indicator_data
    WHERE
        indicator_code = 'RQ.PER.RNK'
        AND year = 2000
        AND indicator_value > 95
);
```



	country
1	Denmark
2	Finland
3	Hong Kong SAR, China
4	Ireland
5	Luxembourg
6	Netherlands
7	Singapore
8	Switzerland
9	United Kingdom
10	United States

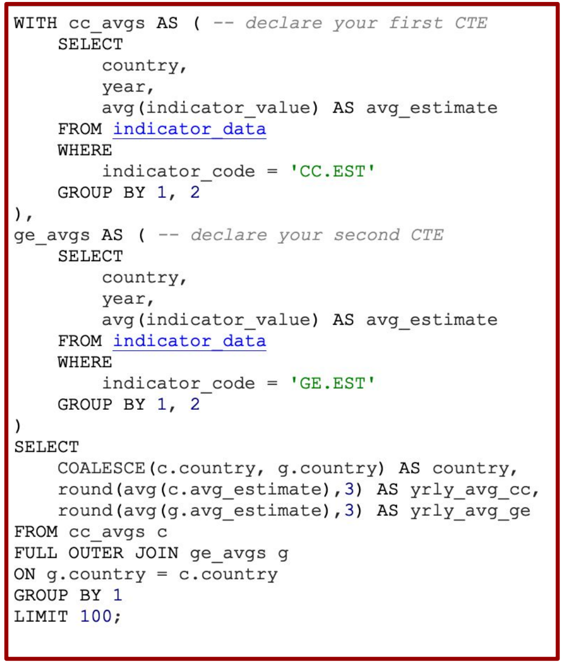
Exercise 7 (5 minutes)

1. Find the average power generation per year for ONLY countries that have geothermal power plants. Complete this using a subquery.
2. Return average power generation by country with a value higher than the ***overall*** average.

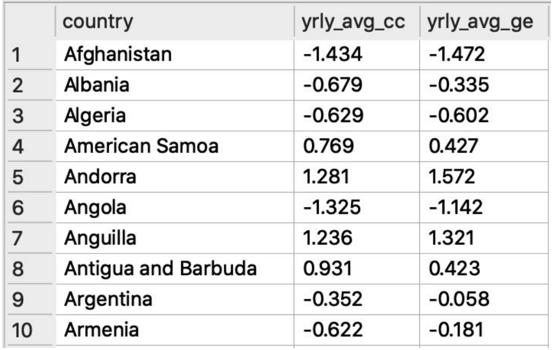
Common Table Expressions

Common Table Expressions (CTEs) are temporary result sets you can reference in a later SQL statement.

- Similar to subqueries in FROM, with more dynamic capabilities
- Can be referenced in other CTEs or the outer query
- Organizes a variety of complex transformations and calculations



```sql
WITH cc_avgs AS ( -- declare your first CTE
    SELECT
        country,
        year,
        avg(indicator_value) AS avg_estimate
    FROM indicator_data
    WHERE
        indicator_code = 'CC.EST'
    GROUP BY 1, 2
),
ge_avgs AS ( -- declare your second CTE
    SELECT
        country,
        year,
        avg(indicator_value) AS avg_estimate
    FROM indicator_data
    WHERE
        indicator_code = 'GE.EST'
    GROUP BY 1, 2
)
SELECT
    COALESCE(c.country, g.country) AS country,
    round(avg(c.avg_estimate),3) AS yrly_avg_cc,
    round(avg(g.avg_estimate),3) AS yrly_avg_ge
FROM cc_avgs c
FULL OUTER JOIN ge_avgs g
ON g.country = c.country
GROUP BY 1
LIMIT 100;
```



	country	yrly_avg_cc	yrly_avg_ge
1	Afghanistan	-1.434	-1.472
2	Albania	-0.679	-0.335
3	Algeria	-0.629	-0.602
4	American Samoa	0.769	0.427
5	Andorra	1.281	1.572
6	Angola	-1.325	-1.142
7	Anguilla	1.236	1.321
8	Antigua and Barbuda	0.931	0.423
9	Argentina	-0.352	-0.058
10	Armenia	-0.622	-0.181

Exercise 8 (5 minutes)

1. Rewrite your first query from Exercise 6 using a Common Table Expression (CTE).

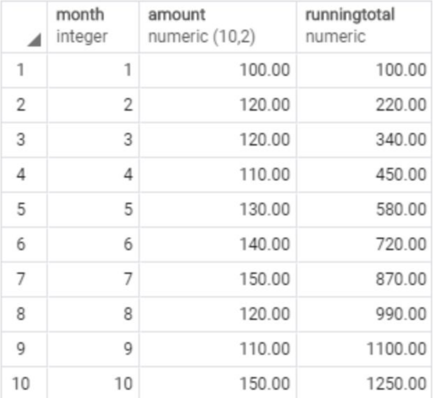
2. Add a second CTE calculating the overall average power generation per year (without country). Subtract the country's value from this overall value to get the difference from the average.

Window Functions

Window functions allow you to make calculations on the ***final result set*** (window).

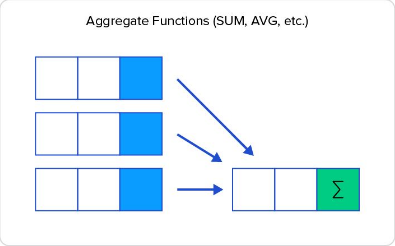
- SQL selects, aggregates, groups, and filters your data
- A window function is used to calculate information based on that final set of information.
- Helpful for overall averages, running totals, ranks, row numbers, etc!

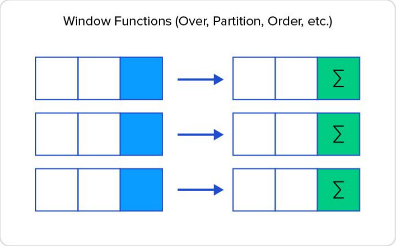
```
SELECT
    month,
    amount,
    SUM(amount) OVER (
        ORDER BY month
    ) RunningTotal
FROM
    SalesInfo;
```



	month integer	amount numeric (10,2)	runningtotal numeric
1	1	100.00	100.00
2	2	120.00	220.00
3	3	120.00	340.00
4	4	110.00	450.00
5	5	130.00	580.00
6	6	140.00	720.00
7	7	150.00	870.00
8	8	120.00	990.00
9	9	110.00	1100.00
10	10	150.00	1250.00

**No exercise on window functions in this section.*





Wrapping up Subqueries

Subqueries and window functions are key to producing a final result set containing the data you need.

- CTEs in particular are ***extremely versatile*** and require practice/testing queries to determine how to best use them.
- Dozens of window functions to calculate row numbers, ranks, previous values, future values, etc!

10 Minute Break

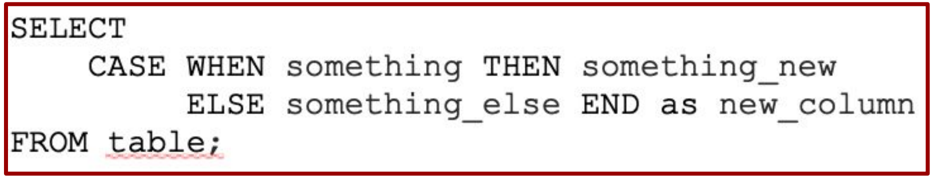
3

Transformations for Analysis

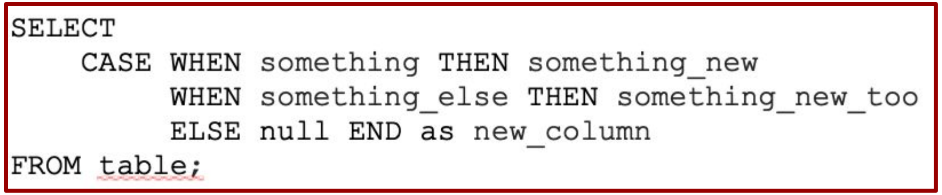
- Conditional (CASE) statements
- Answering business questions
- Prepare a dataset for analysis/machine learning!

Conditional Statements

- SQL's **`CASE`** is similar to the common **`IF/ELSE`** statements
  - Can have multiple **`WHEN`** statements, which are evaluated *sequentially*
  - Can have complex **`WHERE`** conditions with **`AND`**/**`OR`** statements
  - **`ELSE NULL`** is the default final condition, and does not need to be explicitly stated
- ***Tip:*** You can put an entire **`CASE`** statement inside an aggregate function.



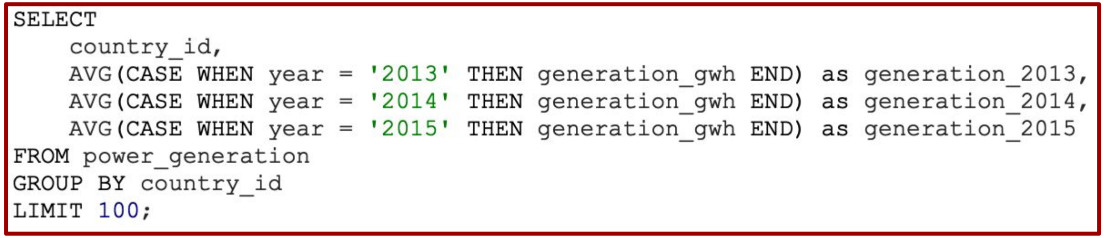
```
SELECT
    CASE WHEN something THEN something_new
         ELSE something_else END as new_column
FROM table;
```



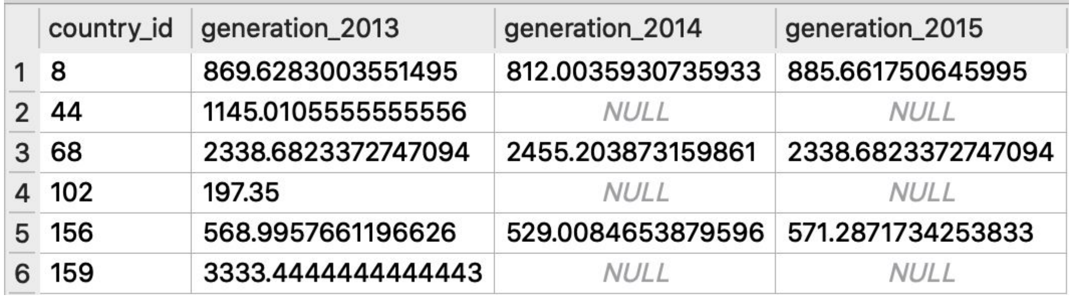
```
SELECT
    CASE WHEN something THEN something_new
         WHEN something_else THEN something_new_too
         ELSE null END as new_column
FROM table;
```

Conditional Statements

- Create new columns without altering the database
- Filter data without using a **`WHERE`** clause
- Pivot data into columns for summary tables, analysis, or machine learning features



```
SELECT
    country_id,
    AVG(CASE WHEN year = '2013' THEN generation_gwh END) as generation_2013,
    AVG(CASE WHEN year = '2014' THEN generation_gwh END) as generation_2014,
    AVG(CASE WHEN year = '2015' THEN generation_gwh END) as generation_2015
FROM power_generation
GROUP BY country_id
LIMIT 100;
```



	country_id	generation_2013	generation_2014	generation_2015
1	8	869.6283003551495	812.0035930735933	885.661750645995
2	44	1145.0105555555556	NULL	NULL
3	68	2338.6823372747094	2455.203873159861	2338.6823372747094
4	102	197.35	NULL	NULL
5	156	568.9957661196626	529.0084653879596	571.2871734253833
6	159	3333.4444444444443	NULL	NULL

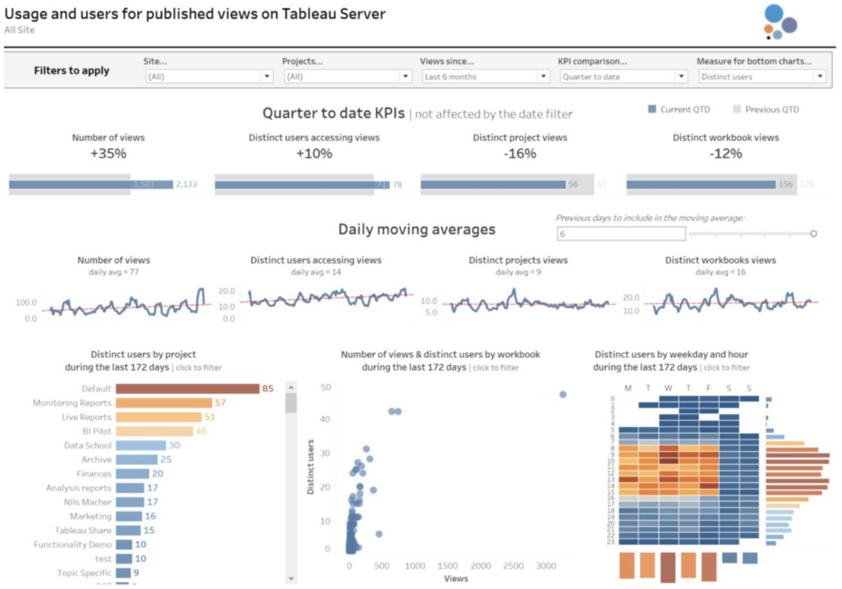
Exercise 9 (5 minutes)

1. How many power plants in the database have a primary fuel that is a form of renewable energy -- `IN ('Hydro', 'Wind', 'Solar', 'Geothermal', 'Wave and Tidal')`?

2. There are a lot of missing values in the `year_of_capacity_data` column. Replace all missing values with the year `'2018'` using a `CASE` statement.

Answering Business Questions

- Stakeholder needs (dashboards, reports, etc.) and analytical projects guide a large proportion of queries and their underlying structure
    - Dictates which SQL skills you’ll leverage most heavily
    - ***Example:*** Mode Analytics as a Business Intelligence tool and to track experiment results
        - A lot of **CASE** statements and subqueries



Usage and users for published views on Tableau Server

All Site

Filters to apply

Site... (All)

Projects... (All)

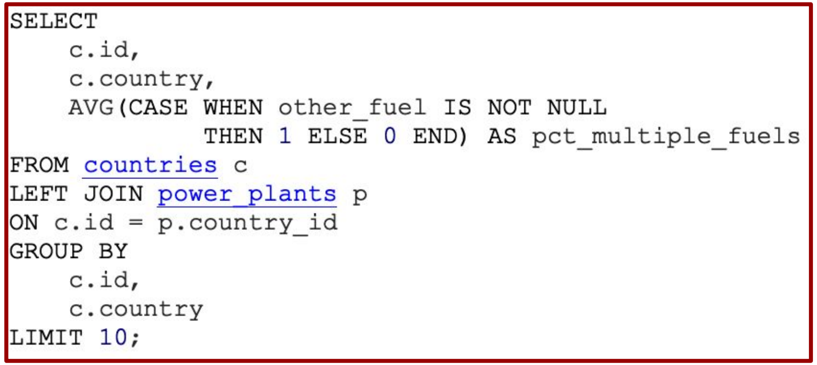
Views since... Last 6 months

KPI comparison... Quarter to date

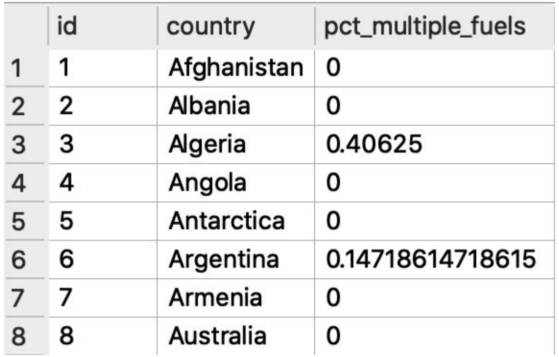
Measure for bottom charts... Distinct users

Answering Business Questions

- How well is an individual performing compared to a benchmark?
- How much has your key metric changed in the past 6 months?
- What's the 3 month rolling average of profit?



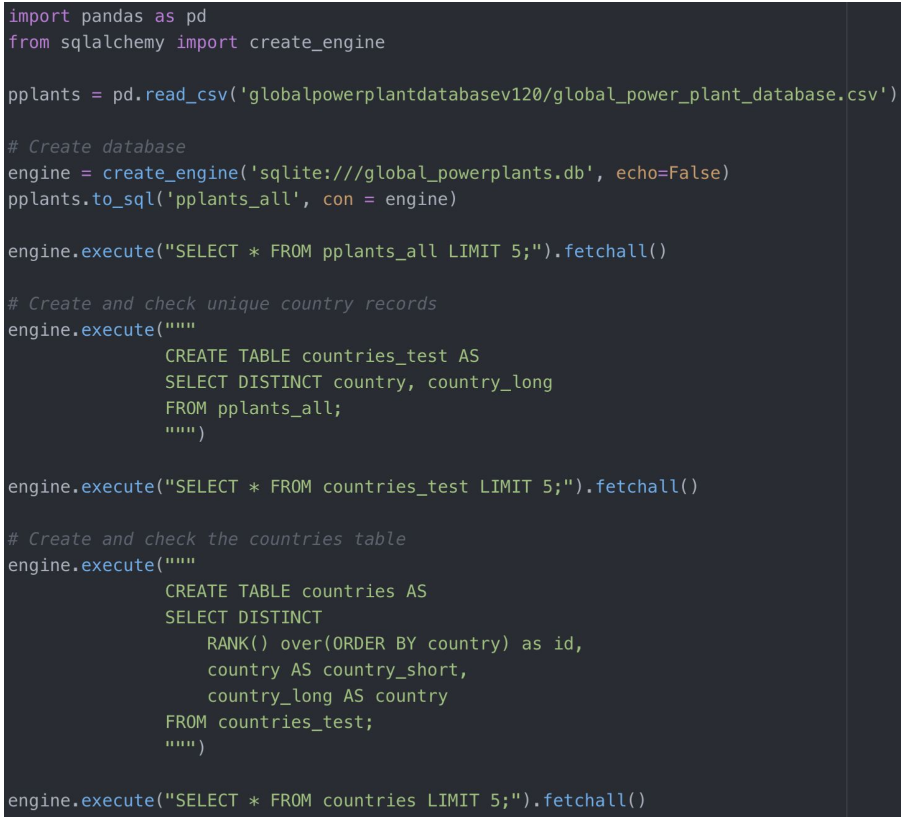
```
SELECT
    c.id,
    c.country,
    AVG(CASE WHEN other_fuel IS NOT NULL
             THEN 1 ELSE 0 END) AS pct_multiple_fuels
FROM countries c
LEFT JOIN power_plants p
ON c.id = p.country_id
GROUP BY
    c.id,
    c.country
LIMIT 10;
```



	id	country	pct_multiple_fuels
1	1	Afghanistan	0
2	2	Albania	0
3	3	Algeria	0.40625
4	4	Angola	0
5	5	Antarctica	0
6	6	Argentina	0.14718614718615
7	7	Armenia	0
8	8	Australia	0

SQL in the Real World

- Prepare SQL queries for further analysis in R/Python/etc
    - Get data in a shape necessary for machine learning or statistical tests
- Run SQL queries directly in R, Python, or a BI tool



```python
import pandas as pd
from sqlalchemy import create_engine

pplants = pd.read_csv('globalpowerplantdatabasev120/global_power_plant_database.csv')

# Create database
engine = create_engine('sqlite:///global_powerplants.db', echo=False)
pplants.to_sql('pplants_all', con = engine)

engine.execute("SELECT * FROM pplants_all LIMIT 5;").fetchall()

# Create and check unique country records
engine.execute("""
               CREATE TABLE countries_test AS
               SELECT DISTINCT country, country_long
               FROM pplants_all;
               """)

engine.execute("SELECT * FROM countries_test LIMIT 5;").fetchall()

# Create and check the countries table
engine.execute("""
               CREATE TABLE countries AS
               SELECT DISTINCT
                   RANK() over(ORDER BY country) as id,
                   country AS country_short,
                   country_long AS country
               FROM countries_test;
               """)

engine.execute("SELECT * FROM countries LIMIT 5;").fetchall()
```

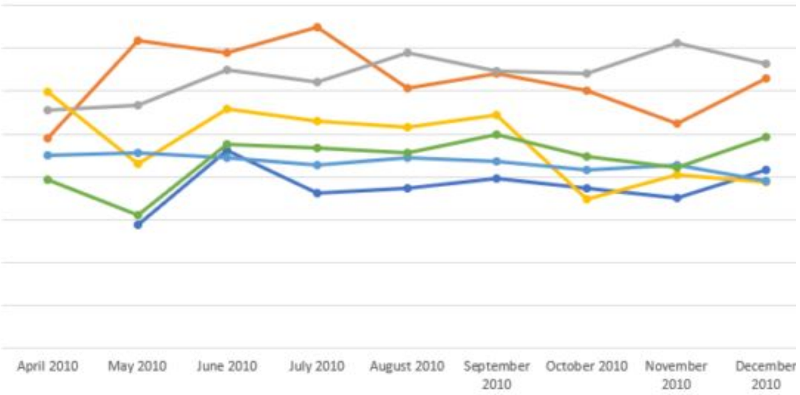
Exercise 10 (5 minutes)

1. Prepare a “pivot table” of countries’ average **`capacity_mw`** by whether or not the power plant’s primary fuel is a renewable **`('Hydro', 'Wind', 'Solar', 'Geothermal', 'Wave and Tidal')`**. Each renewable type should be a separate column.

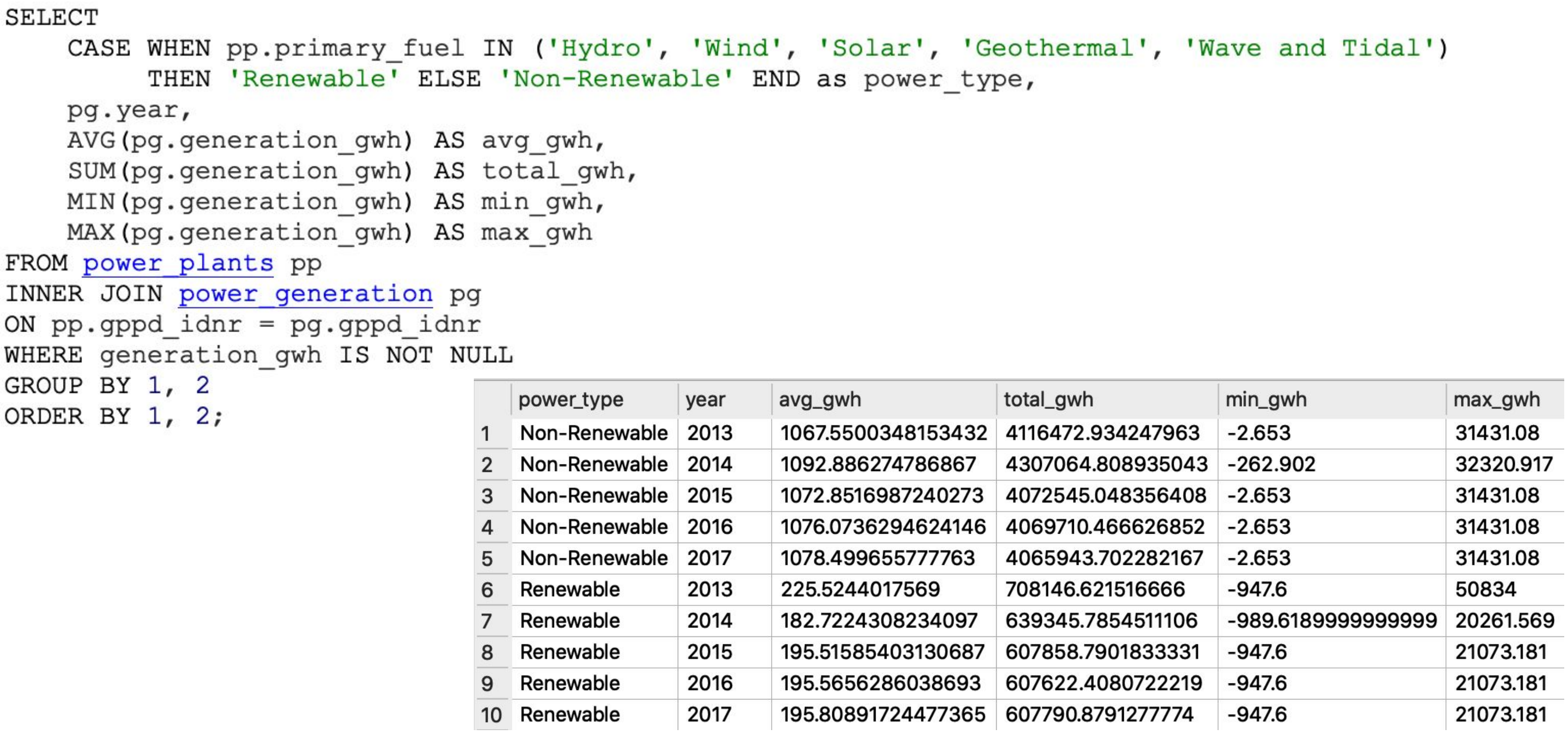
2. Determine the average **`capacity_mw`** across all power plants. Then, calculate the average **`capacity_mw`** by **`primary_fuel`**. What % higher or lower is each fuel type capacity compared to your benchmark average?

Analytics Project (10 minutes)

- Prepare a query for **time series analysis** or **machine learning** answering the following question:
    - What factors might predict the percentage of renewable generation used in a **country** and **year**? Identify 4 to 5 potential predictors of the **% of renewable generation**.
- Things to consider:
    - What do we do with null values?
    - What values do you want represented in separate columns (e.g., needing a CASE statement to represent properly)?
- ***Note:** SQLite does not have a function for calculating the standard deviation, while many other forms of SQL do.



Analytics Project

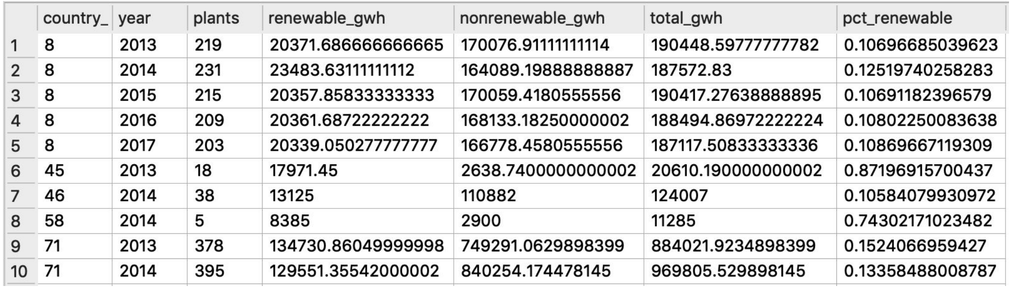


```
SELECT
    CASE WHEN pp.primary_fuel IN ('Hydro', 'Wind', 'Solar', 'Geothermal', 'Wave and Tidal')
         THEN 'Renewable' ELSE 'Non-Renewable' END as power_type,
    pg.year,
    AVG(pg.generation_gwh) AS avg_gwh,
    SUM(pg.generation_gwh) AS total_gwh,
    MIN(pg.generation_gwh) AS min_gwh,
    MAX(pg.generation_gwh) AS max_gwh
FROM power_plants pp
INNER JOIN power_generation pg
ON pp.gppd_idnr = pg.gppd_idnr
WHERE generation_gwh IS NOT NULL
GROUP BY 1, 2
ORDER BY 1, 2;
```

	power_type	year	avg_gwh	total_gwh	min_gwh	max_gwh
1	Non-Renewable	2013	1067.5500348153432	4116472.934247963	-2.653	31431.08
2	Non-Renewable	2014	1092.886274786867	4307064.808935043	-262.902	32320.917
3	Non-Renewable	2015	1072.8516987240273	4072545.048356408	-2.653	31431.08
4	Non-Renewable	2016	1076.0736294624146	4069710.466626852	-2.653	31431.08
5	Non-Renewable	2017	1078.499655777763	4065943.702282167	-2.653	31431.08
6	Renewable	2013	225.5244017569	708146.621516666	-947.6	50834
7	Renewable	2014	182.7224308234097	639345.7854511106	-989.6189999999999	20261.569
8	Renewable	2015	195.51585403130687	607858.7901833331	-947.6	21073.181
9	Renewable	2016	195.5656286038693	607622.4080722219	-947.6	21073.181
10	Renewable	2017	195.80891724477365	607790.8791277774	-947.6	21073.181

Analytics Project Example Query

```sql
WITH power_gen AS (
    SELECT
        pp.country_id,
        pg.year,
        COUNT(pp.gppd_idnr) AS plants,
        SUM(CASE WHEN pp.primary_fuel IN ('Hydro', 'Wind', 'Solar', 'Geothermal', 'Wave and Tidal')
                 THEN pg.generation_gwh END) as renewable_gwh,
        SUM(CASE WHEN pp.primary_fuel NOT IN ('Hydro', 'Wind', 'Solar', 'Geothermal', 'Wave and Tidal')
                 THEN pg.generation_gwh END) as nonrenewable_gwh
    FROM power_plants pp
    INNER JOIN power_generation pg
    ON pp.gppd_idnr = pg.gppd_idnr
    WHERE generation_gwh IS NOT NULL
    GROUP BY 1, 2
)
SELECT
    country_id,
    year,
    plants,
    renewable_gwh,
    nonrenewable_gwh,
    renewable_gwh + nonrenewable_gwh AS total_gwh,
    renewable_gwh / (renewable_gwh + nonrenewable_gwh) AS pct_renewable
FROM power_gen
LIMIT 100;
```



	country_	year	plants	renewable_gwh	nonrenewable_gwh	total_gwh	pct_renewable
1	8	2013	219	20371.686666666665	170076.91111111114	190448.59777777782	0.10696685039623
2	8	2014	231	23483.63111111112	164089.19888888887	187572.83	0.12519740258283
3	8	2015	215	20357.85833333333	170059.4180555556	190417.27638888895	0.10691182396579
4	8	2016	209	20361.68722222222	168133.18250000002	188494.86972222224	0.10802250083638
5	8	2017	203	20339.050277777777	166778.4580555556	187117.50833333336	0.10869667119309
6	45	2013	18	17971.45	2638.7400000000002	20610.190000000002	0.87196915700437
7	46	2014	38	13125	110882	124007	0.10584079930972
8	58	2014	5	8385	2900	11285	0.74302171023482
9	71	2013	378	134730.86049999998	749291.0629898399	884021.9234898399	0.1524066959427
10	71	2014	395	129551.35542000002	840254.174478145	969805.529898145	0.13358488008787

Next Steps to Learn

- Topics
  - Query processing order
  - Improving query performance time
  - Advanced window functions and string manipulation
- Additional databases for practice
  - European Soccer Database
  - NYC Jobs

Questions?

Thank you!

- All materials (slides, exercises) will be uploaded to github.com/mona-kay/aiplus_sql_bootcamp
- Questions? Follow me on twitter at mona_kay_