CO^2 EMISSIONS BY COUNTRY

Data analysis using PostgreSQL and Power BI

TECHNICAL DOCUMENTATION

Monitoring and reducing CO2 emissions is crucial for mitigating climate change and protecting ecosystems. This analysis helps assess emissions by country, supporting informed policy decisions for a sustainable future.

Ricardo Fonseca
github.com/ricardolfonseca

Table of Contents

Choosing a Dataset

The first step was choosing a Dataset from a public repository. I chose **CO2 Emission by countries Year wise (1750-2022)** from **Kaggle.com**

CO2 Emission by countries Year wise (1750-2022)

Creating a Relational Database and Fact Table in PostgreSQL

PostegreSQL allows you to create a database from your personal computer very easily. I created a database named **co2emissions_by_country**.

Then I created a table called **fact_emissions** with the fields from the .csv file (dataset) and their corresponding data types.

```
CREATE TABLE fact_emissions (
    country VARCHAR(80),
    code VARCHAR(80),
    calling_code VARCHAR(80),
    year INT,
    co2emissions FLOAT,
    population2022 INT,
    area INT,
    area_percent_of_world VARCHAR(80),
    density VARCHAR(80)
);
```

As most of this data that will be inserted into the table is not cleaned, some fields like density are not numerical (yet). This will be altered once the data is properly cleaned.

Inserting values into the Fact table

Okay, so PostegreSQL has an amazing feature that can save us a bunch of time, by allowing data to be imported from a .csv file.

Still, I had to use Notepad++ to change some special characters on the original dataset, like ‘²’ and ‘%’. After those replacements I was able to quickly import the data and insert it into the fact table.

By querying the table, we can confirm that the data was imported.
The total count of records imported are 59 619.

```
16  SELECT * FROM fact_emissions LIMIT 10;
17
```

Data Output | Messages | Notifications

	country character varying (80)	code character varying (80)	calling_code character varying (80)	year integer	co2emissions double precision	population2022 integer	area integer	area_percent_of_world character varying (80)	density character varying (80)
1	Afghanistan	AF	93	1751	0	41128771	652230	0.40percent	63/km2
2	Afghanistan	AF	93	1752	0	41128771	652230	0.40percent	63/km2
3	Afghanistan	AF	93	1753	0	41128771	652230	0.40percent	63/km2
4	Afghanistan	AF	93	1754	0	41128771	652230	0.40percent	63/km2
5	Afghanistan	AF	93	1755	0	41128771	652230	0.40percent	63/km2
6	Afghanistan	AF	93	1756	0	41128771	652230	0.40percent	63/km2
7	Afghanistan	AF	93	1757	0	41128771	652230	0.40percent	63/km2
8	Afghanistan	AF	93	1758	0	41128771	652230	0.40percent	63/km2
9	Afghanistan	AF	93	1759	0	41128771	652230	0.40percent	63/km2
10	Afghanistan	AF	93	1760	0	41128771	652230	0.40percent	63/km2

Data cleaning

Now it’s time to select which columns are necessary and which ones should be part of a dimensions table.

Let’s drop the column calling_code as it has non-numerical values.

```sql
-- Drop calling_code
ALTER TABLE fact_emissions
DROP COLUMN calling_code;
```

Next, we’ll convert the column area_percent_of_world to float, but since it has non numerical values we need to remove those characters and then convert it to float.

```sql
-- Clean area_percent_of_world
    -- Step 1: Remove 'percent' and cast the value to float
    UPDATE fact_emissions
    SET area_percent_of_world = CAST(REPLACE(area_percent_of_world, 'percent', '') AS FLOAT);

    -- Step 2: Convert column Data Type to Float
    ALTER TABLE fact_emissions
    ALTER COLUMN area_percent_of_world TYPE FLOAT USING area_percent_of_world::FLOAT;
```

Also, for clarity, we will change the column name area to area_km2:

```sql
-- Clarify area column by changing name to area_km2
    ALTER TABLE fact_emissions
    RENAME COLUMN area TO area_km2;
```

Finally, we will also clean the density column and change its name.

```sql
-- Clean density and change name to population_by_km2_2022
    -- Step 1: Remove '/km2' and cast the value to float
    UPDATE fact_emissions
    SET density = CAST(
        REPLACE(
            REPLACE(density, ',','.'),
        '/km2', '') AS FLOAT);

    -- Step 2: Convert column Data Type to Float
    ALTER TABLE fact_emissions
    ALTER COLUMN density TYPE FLOAT USING density::FLOAT;

    -- Step 3: Change column name
    ALTER TABLE fact_emissions
    RENAME COLUMN density TO population_by_km2_2022;
```

Now there's a clean table 😊 Time to normalize the data.

	country character varying (80)	code character varying (80)	year integer	co2emissions double precision	population2022 integer	area_km2 integer	area_percent_of_world double precision	population_by_km2_2022 double precision
1	Afghanistan	AF	1756	0	41128771	652230	0.4	63
2	Afghanistan	AF	1759	0	41128771	652230	0.4	63
3	Afghanistan	AF	1753	0	41128771	652230	0.4	63
4	Afghanistan	AF	1755	0	41128771	652230	0.4	63
5	Afghanistan	AF	1757	0	41128771	652230	0.4	63
6	Afghanistan	AF	1758	0	41128771	652230	0.4	63
7	Afghanistan	AF	1751	0	41128771	652230	0.4	63
8	Afghanistan	AF	1752	0	41128771	652230	0.4	63
9	Afghanistan	AF	1754	0	41128771	652230	0.4	63
10	Afghanistan	AF	1760	0	41128771	652230	0.4	63

Database Schema and Normalization

Next, we create a dimensions table for the countries, called **dim_country**, which will contain the information needed for the

This table has a PRIMARY KEY – **country_id** - a sequential ID for a unique country.

```sql
-- Create dimension table dim_country
CREATE TABLE dim_country (
    country_id SERIAL PRIMARY KEY,
    country_name VARCHAR(80),
    population2022 INT,
    area_km2 INT,
    area_percent_of_world FLOAT,
    population_by_km2_2022 FLOAT
);
```

Next, let's fill out the unique countries from the fact_emissions table.

```sql
-- Populate with distinct values from fact_emissions
INSERT INTO dim_country (
    country_name,
    population2022,
    area_km2,
    area_percent_of_world,
    population_by_km2_2022
    )
SELECT
    DISTINCT country,
    population2022,
    area_km2,
    area_percent_of_world,
    population_by_km2_2022
FROM fact_emissions;
```

There! Now we have 220 unique countries in the dim_country table.

Now let's get back to our fact table, create the column country_id and reference it as a FOREIGN KEY. After that, we can update the column in the facts table with the respective country_id for each country.

```sql
-- Create column country_id in fact_emissions table
ALTER TABLE fact_emissions
ADD COLUMN country_id INT;

-- Set it as a foreign key to reference dim_country
ALTER TABLE fact_emissions
ADD CONSTRAINT fk_country_id
FOREIGN KEY (country_id)
REFERENCES dim_country(country_id);

-- Update column with the country_id values from dim_country
UPDATE fact_emissions fe
SET country_id = dc.country_id
FROM dim_country dc
WHERE fe.country = dc.country_name;
```

Now we can drop the following columns from the facts table, since that information is already stored in the dimensions table.

```sql
ALTER TABLE fact_emissions
    DROP COLUMN country,
    DROP COLUMN code,
    DROP COLUMN population2022,
    DROP COLUMN area_km2,
    DROP COLUMN area_percent_of_world,
    DROP COLUMN population_by_km2_2022
;
```

Handling Null Values in Dimensions Table (Additional Data Cleaning)

Validating the data in the **dim_country** name we can see that there are some null values in columns population2022, area_km2, population_by_km2_2022 and area_percent_of_world.

After querying for null values in population_2022 or area_km2, we get 31 rows.

```sql
SELECT country_id, country_name
    FROM dim_country
    WHERE population2022 IS NULL OR area_KM2 is null;
```

After retrieving those values manually, we can update our table to math these values by country_id:

```
UPDATE dim_country
SET
    population2022 = CASE country_id
        WHEN 43 THEN 102396968
        WHEN 110 THEN 7925
        WHEN 23 THEN 6825
        WHEN 85 THEN 1843
        WHEN 24 THEN 10693939
        WHEN 61 THEN 54672
        WHEN 68 THEN 1802250
        WHEN 167 THEN 6035103
        WHEN 190 THEN 28010186
```

...

```
END,
area_km2 = CASE country_id
    WHEN 43 THEN 342000
    WHEN 110 THEN 122
    WHEN 23 THEN 32
    WHEN 85 THEN 135
    WHEN 24 THEN 78865
    WHEN 61 THEN 1393
    WHEN 68 THEN 10887
    WHEN 167 THEN 2344858
    WHEN 190 THEN 322463
```

...

```
    END
WHERE country_id IN (
    43, 110, 23, 85, 24, 61, 68, 167, 1
    138, 152, 66, 1, 216, 90, 179, 8, 2
    105, 29, 97
```

As for area_percent_of_world, we can update the values for the countries with null values that can calculate the area percentage by dividing the country area by the total area of the world.

Note that by placing a decimal at the end of the world area, the value returned will also be a decimal, hence the round formula to present a more readable number.

```
UPDATE dim_country
SET area_percent_of_world = round((area_km2 / 510072000.0) * 100,1)
WHERE country_id IN (
    23, 85, 68, 195, 138, 66, 167, 190, 135, 206, 143, 144, 1, 216, 8, 219
);
```

Lastly, let's calculate the population_by_km2_2022 by dividing the population2022 by area_km2.

```
UPDATE dim_country
SET population_by_km2_2022 = ROUND(population2022 / area_km2, 1)
WHERE country_id IN (
    43, 110, 24, 61, 152, 23, 85, 167, 190, 135, 206, 143, 90,
    179, 8, 27, 168, 148, 139, 106, 219, 105, 29, 97
);
```

And that's it! No more pesky null values 😊

Joining Tables (Get continents)

Now, how about if we could add the continent to our dimensions table? Let's do it.

First, I got a .csv list of countries and continents from List of Countries by Continent 2024.

This time I used Excel for a little help in adding the necessary characters to quickly copy and paste the list into the SQL formula:

	country		continent		
('	India	','	Asia	'),	('India',' Asia'),
('	China	','	Asia	'),	('China',' Asia'),
('	United States	','	North America	'),	('United States',' North America'),
('	Indonesia	','	Asia	'),	('Indonesia',' Asia'),
('	Pakistan	','	Asia	'),	('Pakistan',' Asia'),

Now, on to create a new table called continents_table (that will be dropped later).

```
CREATE TABLE continents_table (
    country_name VARCHAR(80),
    continent VARCHAR(80)
);

INSERT INTO continents_table (country_name, continent)
VALUES ('India',' Asia'),
('China',' Asia'),
('United States',' North America'),
('Indonesia',' Asia'),
('Pakistan',' Asia'),
('Nigeria',' Africa'),
('Brazil',' South America'),
```

Let's do a quick query just to make sure if all the countries have a match or if there are any null values.

```
SELECT country_id, dc.country_name
FROM dim_country AS dc
LEFT JOIN continents_table AS ct
ON dc.country_name = ct.country_name
WHERE continent IS NULL;
```

Looks like there are 12 countries without a match. Don't worry, we'll add those manually.

First, let's create the new column continent_name in dim_country and fill it out:

```
ALTER TABLE dim_country
ADD COLUMN continent_name VARCHAR(80);

UPDATE dim_country dc
SET continent_name = ct.continent
FROM continents_table ct
WHERE dc.country_name = ct.country_name;
```

Now on to update those null records:

```
UPDATE dim_country
SET continent_name = 'Africa'
WHERE country_id IN (43, 110, 167, 190);

UPDATE dim_country
SET continent_name = 'Antarctica'
WHERE country_id IN (135);

UPDATE dim_country
SET continent_name = 'Asia'
WHERE country_id IN (23, 85, 206);

UPDATE dim_country
SET continent_name = 'Europe'
WHERE country_id IN (24, 61, 68);

UPDATE dim_country
SET continent_name = 'North America'
WHERE country_id IN (143);
```

Finally, let's drop continents_table since we won't need it anymore.

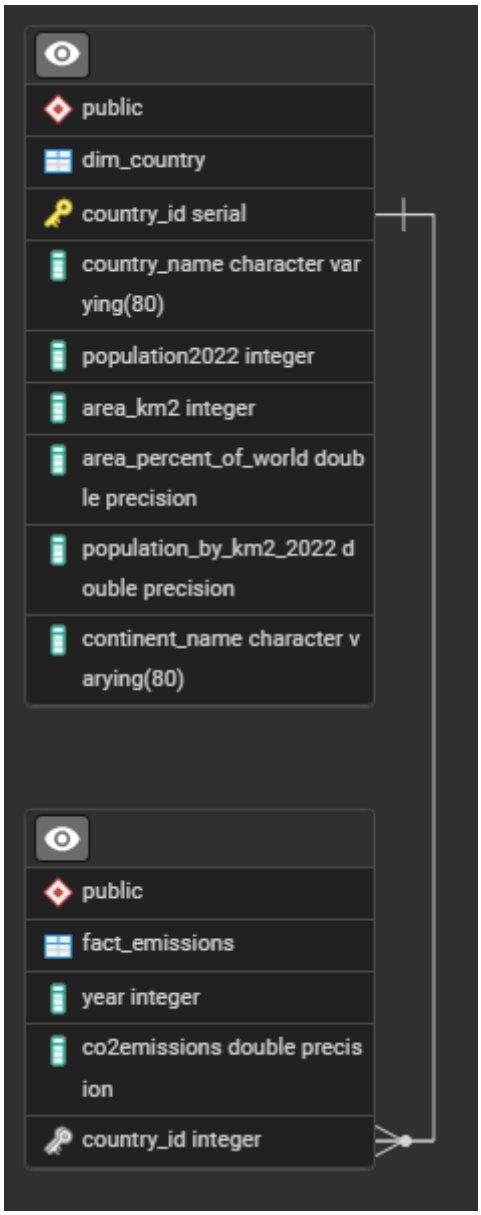
```
DROP TABLE continents_table;
```

And voilá!

Entity Relationship Diagram (ERD)

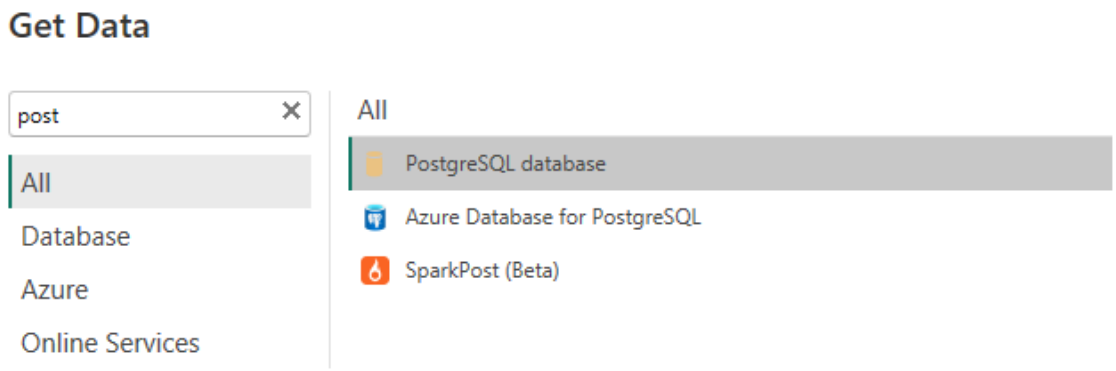
This is the ERD for database **co2emissions_by_country.**

It is now ready to be connected to Power BI for analysis and reporting. 🥳

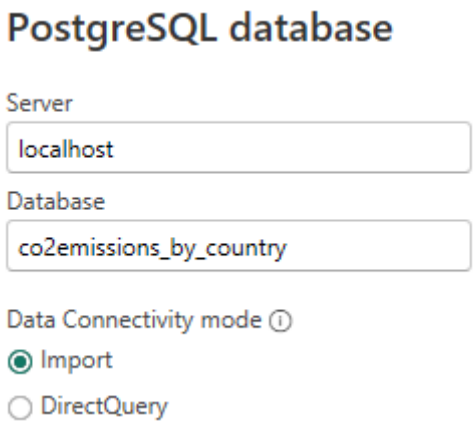


Connecting Power BI to PostgreSQL Database

Power BI allows to connect automatically to PostgreSQL databases.



Since we don't get real time data in this database, Import is more suitable over DirectQuery, which could impact performance for the report.

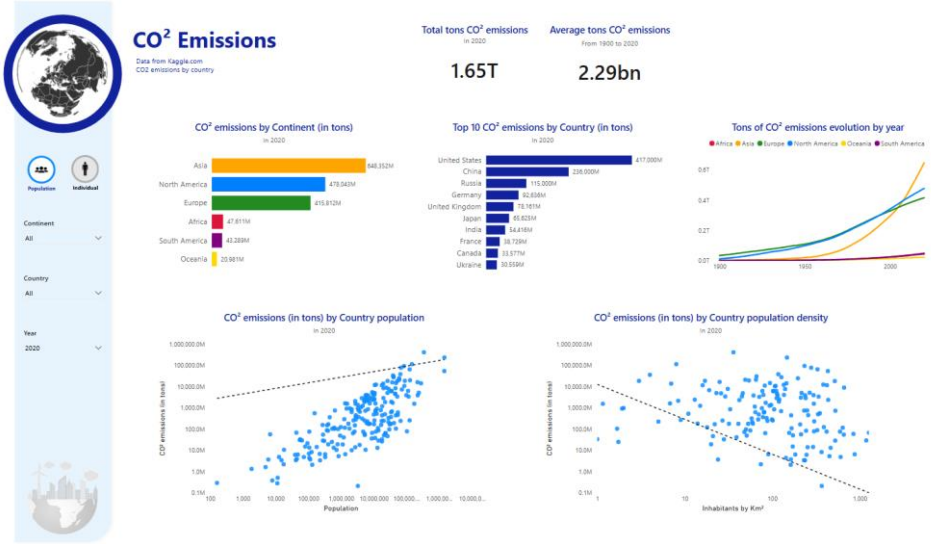


Navigating the dashboard

This dashboard displays the data in a way that makes it easy to get insights from the data.

On the left there are filters that can be applied:

- Population/Individual buttons: toggles between total emissions for total population vs. each country inhabitant
- Continent: multiple selection available
- Country: multiple selection available
- Year: year of emissions. The latest year is presented by default



Notice that the map on top will change accordingly to the region selected.