A new database

In this guided practice we're going to create a database on energy production from scratch with entirely fake data.

The database will hold data about energy production including solar, wind, geothermal, nuclear and fossil (coal, gas, ...)

For instance :

- list of production entities from individual roof top solar panels, to wind farms , coal and gas factories, nuclear facilities
- location: country etc
- life cycle: built date, lifespan, end of life, maintenance
- energy produced
- carbon and other gaseous emissions
- cost, ROI
- daily production

We build the database with the following steps

- create the database on your local machine
- create the tables
- We use a couple of PL/SQL functions to generate the data
- run the functions to generate a high volume of rows.

> **This is Fake data** : The data generated does not claim to represent the reality in terms of CO2 emissions for different types of energy production. The function we use is quite basic and the logic behind all the attributes of facilities is very simple. This does not reflect reality.

Log in to the server

Make sure PostgreSQL is running on your local machine

- on Mac

```bash
brew services info postgres@16
```

- on Windows

```PowerShell
pg_ctl info
```

and if that does not work, this should :

```PowerShell
> & "C:\Program Files\PostgreSQL\16\bin\pg_ctl" start -D "C:\Program Fi
```

Then connect on your local server with

- Mac : `psql -h localhost -d postgres` or with your local username `psql -h localhost -U <username> -d postgres`
- Windows : `psql -h localhost -U postgres -d postgres`

Create the database

In `psql`, create a new database with

```SQL
CREATE DATABASE energydb
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LOCALE_PROVIDER = 'libc'
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;
```

Then connect to the database with

```SQL
\c energydb
```

Create the table structure

```sql
-- Create tables
CREATE TABLE energy_sources (
    id SERIAL PRIMARY KEY,
    source_name VARCHAR(50) NOT NULL
);

CREATE TABLE countries (
    id SERIAL PRIMARY KEY,
    country_name VARCHAR(100) NOT NULL
);

CREATE TABLE production_entities (
    id SERIAL PRIMARY KEY,
    entity_name VARCHAR(100) NOT NULL,
    energy_source_id INTEGER REFERENCES energy_sources(id),
    country_id INTEGER REFERENCES countries(id),
    capacity_mw NUMERIC(10, 2) NOT NULL,
    built_date DATE NOT NULL,
    lifespan_years INTEGER NOT NULL,
    end_of_life_date DATE NOT NULL,
    last_maintenance_date DATE,
    total_energy_produced_mwh NUMERIC(15, 2) NOT NULL,
    carbon_emissions_tons NUMERIC(10, 2),
    other_emissions_tons NUMERIC(10, 2),
    initial_cost_usd NUMERIC(15, 2) NOT NULL,
    roi_percent NUMERIC(5, 2) NOT NULL
);

CREATE TABLE energy_production_daily (
    id SERIAL PRIMARY KEY,
    production_entity_id INTEGER REFERENCES production_entities(id),
    date DATE NOT NULL,
    energy_produced_mwh NUMERIC(10, 2) NOT NULL
);
```

And insert some data into the categorical tables : `countries` and `energy_source` :

```sql
INSERT INTO energy_sources (source_name) VALUES
('Solar'), ('Wind'), ('Geothermal'), ('Coal'), ('Natural Gas'), ('Nucle

INSERT INTO countries (country_name) VALUES
('United States'), ('China'), ('Germany'), ('India'), ('Japan'),
('United Kingdom'), ('France'), ('Italy'), ('Canada'), ('Australia');
```

You can add whatever country you want. But modifying the `energy_sources` data

requires to modify the function below.

The rows in the `energy_production_daily` and `production_entities` tables will be filled using 2 PL/pgSQL functions.

The data generating functions

Just copy paste these functions into your `psql` terminal.

They are written in PL/pgSQL. Something we'll look at next week.

The function `generate_production_entities` generates data for the production entities and `generate_daily_production` daily energy production.

Both functions take the number of rows you want to generate as input.

SQL

```sql
CREATE OR REPLACE FUNCTION generate_production_entities(num_rows INTEGEI
RETURNS VOID AS $$
DECLARE
    i INTEGER;
    v_energy_source_id INTEGER;
    v_country_id INTEGER;
    v_capacity_mw NUMERIC(10, 2);
    v_built_date DATE;
    v_lifespan_years INTEGER;
    v_end_of_life_date DATE;
    v_last_maintenance_date DATE;
    v_total_energy_produced_mwh NUMERIC(15, 2);
    v_carbon_emissions_tons NUMERIC(10, 2);
    v_other_emissions_tons NUMERIC(10, 2);
    v_initial_cost_usd NUMERIC(15, 2);
    v_roi_percent NUMERIC(5, 2);
BEGIN
    FOR i IN 1..num_rows LOOP
        -- Generate random values for each column
        v_energy_source_id := floor(random() * 6 + 1);
        v_country_id := floor(random() * 10 + 1);
        v_capacity_mw := random() * 1000;
        v_built_date := random_date('2000-01-01', '2023-12-31');
        v_lifespan_years := floor(random() * 30 + 10);
        v_end_of_life_date := v_built_date + (v_lifespan_years || ' yea
        v_last_maintenance_date := random_date(v_built_date, LEAST(v_en
        v_total_energy_produced_mwh := v_capacity_mw * 24 * 365 * (EXTR
        v_carbon_emissions_tons := CASE WHEN v_energy_source_id IN (4, 
        v_other_emissions_tons := CASE WHEN v_energy_source_id IN (4, 5
        v_initial_cost_usd := v_capacity_mw * (CASE v_energy_source_id
```

```
                                                WHEN 1 THEN 1000000
                                                WHEN 2 THEN 1500000
                                                WHEN 3 THEN 2000000
                                                WHEN 4 THEN 3000000
                                                WHEN 5 THEN 1000000
                                                WHEN 6 THEN 6000000
                                                END) * random();
        v_roi_percent := random() * 20;

        -- Insert the generated data
        INSERT INTO production_entities (
            entity_name, energy_source_id, country_id, capacity_mw, bui
            end_of_life_date, last_maintenance_date, total_energy_produ
            other_emissions_tons, initial_cost_usd, roi_percent
        ) VALUES (
            'Entity_' || i, v_energy_source_id, v_country_id, v_capacity
            v_end_of_life_date, v_last_maintenance_date, v_total_energy
            v_other_emissions_tons, v_initial_cost_usd, v_roi_percent
        );
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

We also need to create daily energy production for our production facilities.

```SQL
CREATE OR REPLACE FUNCTION public.generate_daily_production(num_days int
 RETURNS void
 LANGUAGE plpgsql
AS $function$
DECLARE
    v_batch_size INTEGER := 1000; -- Adjust this value based on your sy
    v_batches INTEGER := CEIL(num_days::float / v_batch_size);
    v_entity_ids INTEGER[];
    v_entity_capacities NUMERIC[];
    v_max_id INTEGER;
    v_min_id INTEGER;
BEGIN
    -- Get the range of production entity IDs
    SELECT MIN(id), MAX(id) INTO v_min_id, v_max_id FROM production_ent

    -- Precompute random entity IDs and their capacities
    WITH random_entities AS (
        SELECT id, capacity_mw
        FROM production_entities
        WHERE id IN (
            SELECT (random() * (v_max_id - v_min_id) + v_min_id)::integ
            FROM generate_series(1, LEAST(num_days, v_max_id - v_min_id
        )
    )
    SELECT array_agg(id), array_agg(capacity_mw)
    INTO v_entity_ids, v_entity_capacities
    FROM random_entities;

    -- Generate and insert data in batches
    FOR i IN 1..v_batches LOOP
        INSERT INTO energy_production_daily (production_entity_id, date
        SELECT
            v_entity_ids[1 + ((gs.id - 1) % array_length(v_entity_ids,
            CURRENT_DATE - (random() * 365)::INTEGER,
            random() * v_entity_capacities[1 + ((gs.id - 1) % array_len
        FROM generate_series(1, LEAST(v_batch_size, num_days - (i-1)*v_
    END LOOP;
END;
$function$;
```

Generate the data

We can generate the actual data and fill out the tables by calling these functions with a number of rows as input.

- Generate 10,000 facilities with

```sql
-- Generate 1 million rows
SELECT generate_production_entities(10000);
```

- Then generate 50,000 rows of daily production with

```sql
-- Generate 500k rows of daily production data
SELECT generate_daily_production(500000);
```

You may want to try with fewer rows to begin with and make sure everything is working as expected.

For instance, 100 entities and 1000 daily production

```sql
SELECT generate_production_entities(100);
SELECT generate_daily_production(1000);
```

Bake for 20 minutes

This should take some time to run. We will use that database later on for a practice on Indexes. So launch the data production and in the meantime let's go back to Indexes.