Max Smith
CSPB3287 - Kim

Energy Scout – Final Project Report

Project Links:

Github

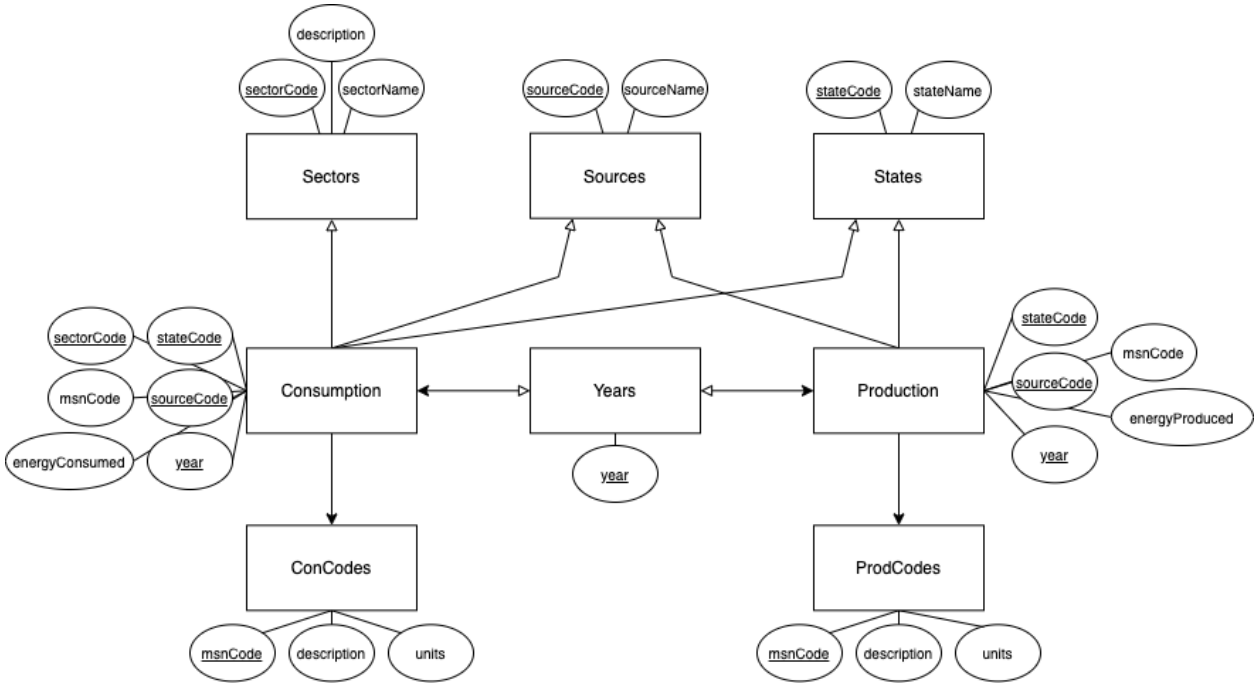
YouTube

Concept

Energy Scout is a database derived from EIA.gov data, built for purposes of performing broad analysis of energy production and consumption in the United States. It allows users to analyze Energy Produced and Energy Consumed in a standard unit of Billions BTU across all states, different energy source, and end-use sectors. The data goes back to 1960 and up to 2018. By providing a bird's eye view of energy production and consumption, users can gain an understanding of general production and consumption patterns, which provides an entry point into more interesting questions and insights that can be explored by either running queries against the Energy Scout database or by delving deeper into the EIA.gov data, which has many, many more subcategories.

Data

The EIA.gov data has a lot of depth and texture, but it can be difficult to navigate and gain a broad sense of since the summary codes are not differentiated from the other categorical codes and the dat is not organized in a way that makes it directly accessible. More recently they have released the S.E.D.S data, which provides the same nationwide data but on a state by state basis. They also have a visualization tool in Beta for exploring the data, but users are limited by its UI functionality. By downloading the production and consumption data, identifying the most salient, summary codes by reading through the reference documentation and indexes, then filtering out the codes I didn't want and finally reorganizing the data into a relational format that can be queried and explored, I have hopefully provided users direct access to these more generic tasks without them having to navigate the website and piece the information together bit by bit. The final relational diagram I settled on is as follows:



This design allow for the most flexibility while doing its best to reduce data redundancy. Attributes can be added to the Sectors, Sources, States, and Years relations without concern for growing the Consumption and Production relations, which are by far the largest tables. These tables are related using foreign keys that together form a primary key, since all of these other relations are required for identifying each separate energyProduced or energyConsumed value. While not necessary, I included the ConCodes and ProdCodes tables since these retain the energy codes used by the EIA.gov website, which I felt was important because the full MSN code is associated with a description and unit, which means the data source can be traced back to the original source for more context while also allowing for additional units to be added in the future if that was desired. The MSN code is a required field for this reason, but is not a part of the primary key since the MSN code itself is just an aggregate of the state, sector, and source codes. The foreign key codes themselves are only two characters, which means they take up little room but also allow for joins to be made for more advanced queries or even just acquiring the more readable names they represent, such as full state names.

Tools and Deployment

The data I downloaded from the EIA.gov website came in Excel files. I manually created several codes files for reference, then I used Python in a Jupyter notebook to import the data into a pandas library dataframe, where I can filter, rebuild, rename, and finally export the data into its final relational format as a csv file. I have included a pdf of the notebook in my git repo for easy reference of the code. For my database I used PostgreSQL 12. I originally planned to host it on Heroku, however their free tier had a 10k row limit which I ended up exceeding by far. Instead, I created a project on Google Cloud Platform

and, using my remaining school credits to run it, I provisioned a Postgres instance to host my database. I then used pgAdmin, a Postgres tool similar to MySQL Workbench, to connect to the Postgres instance server and define the relations in my database. After defining the tables, columns, and constraints, I also used pgAdmin to import the csv files into my database. I then used pgAdmin to run queries on my tables to verify the data and relationships were correct. Finally, I deployed a copy of Metabase to my Heroku account using their one-click deployment. I then connected the Metabase instance to my database. In order to secure the connection, I would need to enable SSL on the database instance and whitelist the requesting IP address. Because Heroku dynos are hosted on Amazon ECS their originating IP addresses are fairly dynamic, so I would need to use a proxy add on such as Fixie Socks, but unfortunately Metabase does not support configuring traffic over a proxy. The solution would be to host the database on Heroku or to host Metabase on GCP Kubernetes, but costs would be associated in either case and so for the purposes of this project only I simply whitelisted all incoming IP addresses to the database and did not secure the connection.

Learned goals and potential improvements

As expected, I learned quite a bit about the energy production and consumption patterns of the different states. This was necessary to being able to design a database. How the database is designed plays a major role in how it can be interpreted. I learned that reformatting, naming, and structuring the data in order to prepare it for a database can be a major step. By using Python to programmatically handle these steps I realized that I could use the EIA.gov API to dynamically clean and add data to my database. To do this, however, one would need to be very wary about checking for all kinds of unexpected items that could crop up in the data being imported, which makes this step fairly daunting. I applied the learning from our class to designing a database using a new database and design tool, Postgres and pgAdmin. These definitely brought some new challenges, so I was forced to keep my design somewhat simple to avoid having issues I couldn't correct in a timely manner. I would like to go back after interacting with the database some more and make some changes to make it more complete. For example I would like to make some inherited tables for Production and Consumption that are 10 year subsets of the data for faster and easier querying, as well as updating some constraints such as setting the year table deletions to cascade. I would also like to add primary indexes to the Production and Consumption tables, since inserts on these tables are infrequent, they are the largest tables, and the most often queried tables. I learned some of the challenges involved with deploying an app that interfaces with a database as well. I would like to re-deploy Metabase securely by deploying it to Kubernetes using Terraform or by hosting the database on Heroku. I think for something this simple Heroku would be best, but I also learned how much more configurable Google Cloud Platform was which I think I would prefer for longer term projects, since it allows for more control.