ChatDB: Using Natural Language Processing to Query PostgreSQL Databases

Kevin O'Kane

Github Link: https://github.com/krokane/chatDB

Introduction

ChatDB is a program designed to replicate simple tasks of the ChatGPT chatbot, specifically in regards to a dataset inserted into the program. The program is a beginner's tool to understanding SQL querying of a database, allowing for five main functionalities – uploading a data table to a PostgreSQL database, understanding a data table through looking at metadata, looking at example SQL queries of that table to understand basic SQL clauses, asking natural language questions of that data table, and ultimately using the provided SQL queries to pull specific data from the database. In the example implementation of ChatDB, these functionalities are accomplished through an interface built into the command line of a local machine. For this study, three main datasets were used as examples for the program. The first dataset is an example coffee shop sales database, representing transactions for a coffee shop. The second two are datasets that come from the Museum of Modern Art, in New York City. The datasets, *artists* and *artworks,* represent records of all the artwork in MOMA. Ultimately, the program is a simple interface with the goal of teaching very basic SQL queries to an intro-level user. It shows what different clauses in a SQL query represent in final data sets, as well as how everyday questions can be turned into these types of queries.

Planned Implementation

Without the capability to deploy machine learning + natural language processing toolkits, the biggest hurdle for this work was always being able to take in natural language questions, process them, and produce a relevant SQL query for the user to run and gain a relevant answer. My original project proposal notes that this will be hard on two aspects of a natural language query – both identifying key words that trigger accurate SQL clauses (i.e. group by, where, etc.) and identifying keywords that reference variables both directly and indirectly. The examples of being able to know that price times quantity equals revenue or revenue minus cost equals profit are microcosms of the largest problem at hand. Without including a reference to a specific column in the data table, how can the program know that when asking for profit they need two values within the data table multiplied by each other. This was noted in my midterm proposal as well as being the biggest hurdle to success in the project.

My initial answer to this problem was to create SQL templates that can be deployed to match these words to queries. The idea would be that in the natural language question, the only words the program would need is a trigger to identify the correct template (i.e. something that matches to the correct SQL clause) and then something that could match to the correct variables requested for the query. However, this strategy proved to be labor intensive. Creating dozens of templates would be a lot of work, and then the logic to match these templates to a query would likely result in lots of incorrect queries.

Instead of creating all these templates, the strategy changed to more of a keyword matching system. The goal would be to eliminate all unnecessary words from the natural language question using non-ML NLP tools from NLTK. Once the question was boiled down to a simple few words, the goal would be to match each word to either a relevant SQL clause or variable name. The matching logic would use a built out vocabulary, which would be searched through to figure out what word means what. This allowed for simple and complex queries to be built with different types of querying clauses all in one query. While the vocabulary would be the biggest hurdle to overcome without ML tools, it would be much less labor-intensive to build a vocabulary of matching words than to build a library of SQL queries to match to.

Flow Diagram

1. To run the program, enter "python chatDB.py" into your command line.
2. The program will start by giving you five choices of what to run. It will ask for an input corresponding to the task you would like to complete
3. If your input = 1:
    a. You have selected the ask a natural language question option. You will be prompted to provide the system with a question. It will note that in order to filter by a number, you must include a "#" in your question.
    b. The question will be entered into the sql_list_to_query function, which will process the natural language question into a SQL query
    c. The newly created SQL query will be inserted into the run_sql_query function, which will form a dataframe
    d. The dataframe will be returned to the user to answer their question with data
4. If your input = 2:
    a. You have selected the random query generator
    b. The program will print the data tables currently in the PostgreSQL database, and then the user can return an input of which data table they'd like their example queries from

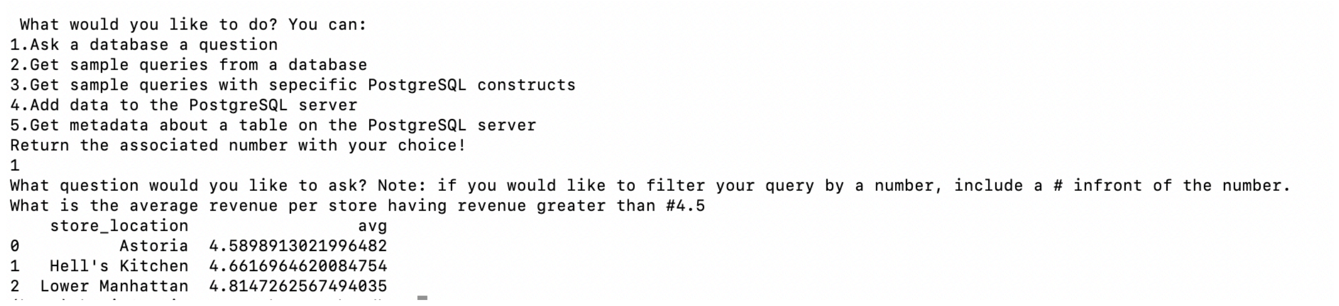
    c. The program will run the get_random_sql function, which produces three random SQL queries. The program then asks the user to input a number, 1, 2, or 3, corresponding to the SQL query they would like to run
    d. Once selected, the query is inserted into the run_sql_query function, which returns a dataframe of the data from the selected query
5. If your input = 3:
    a. You have selected the query generator for specific SQL clauses
    b. The program will print the data tables currently in the PostgreSQL database, and then the user can return an input of which data table they'd like their example queries from
    c. The program will again prompt the user, this time for a number associated with a specific SQL clause type (where, group by, having, order by, or limit)
    d. The chosen construct and data are imputed into the gen_sql_query function, which returns SQL queries fitting that construct. The program then asks the user to input a number, 1, 2, or 3, corresponding to the SQL query they would like to run
    e. Once selected, the query is inserted into the run_sql_query function, which returns a dataframe of the data from the selected query
6. If your input = 4:
    a. You have chosen the file upload choice
    b. The program will ask for a file path to your dataset that you'd like to insert into the PostgreSQL database, which will be inserted into the upload_file_postgres function, which uploads the data to the server
7. If your input = 5:
    a. You have selected the metadata option
    b. The program will print the data tables currently in the PostgreSQL database, and then the user can return an input of which data table they'd like to explore metadata from
    c. Once selected, the get_metadata function will return the table name, column names, and a brief sample of the dataset

Implementation

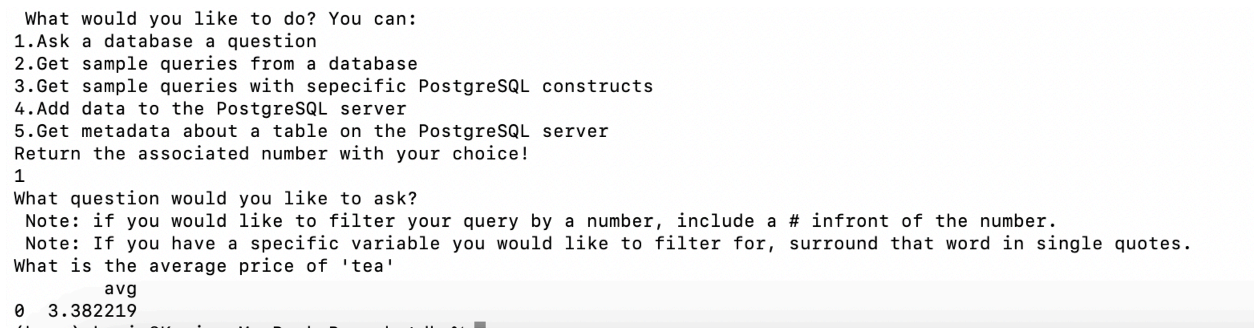
The program is utilized through the command line, as mentioned above. It requires Python to run the program, as well as a local PostgreSQL server. The packages used in python for the program include *argparse*, *logging*, *nltk, pandas*, *psycopg2, os, random, re, sys, time, defaultdict,* and *sqlalchemy.* Of note, the NLTK package allows for a variety of natural language processing tools that allow for the keyword matching algorithm to work smoothly. Also, the Psycopg2 and SQLAlchemy packages allow for the Python script to run SQL queries to the PostgreSQL server.

Here are a few examples of the program running smoothly from the command line:

Natural Language Question Answering

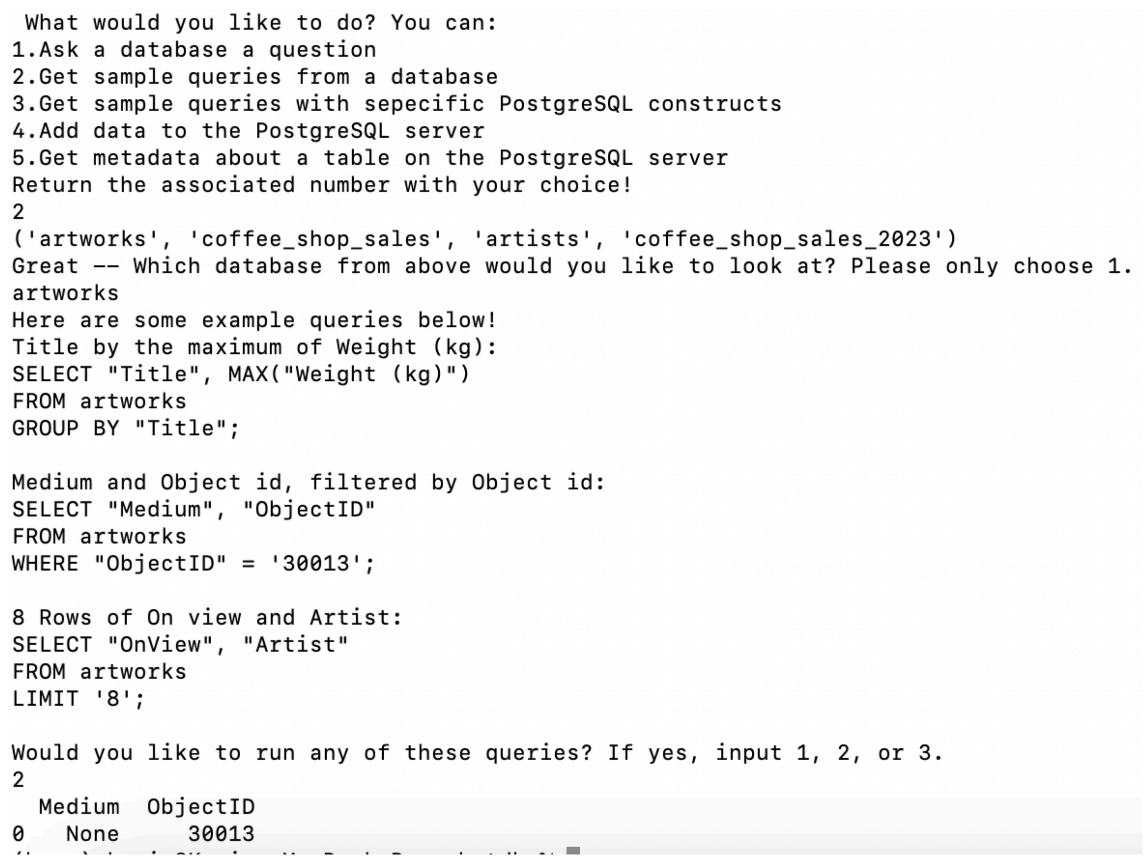


```
 What would you like to do? You can:
1.Ask a database a question
2.Get sample queries from a database
3.Get sample queries with sepecific PostgreSQL constructs
4.Add data to the PostgreSQL server
5.Get metadata about a table on the PostgreSQL server
Return the associated number with your choice!
1
What question would you like to ask? Note: if you would like to filter your query by a number, include a # infront of the number.
What is the average revenue per store having revenue greater than #4.5
    store_location                avg
0          Astoria  4.5898913021996482
1   Hell's Kitchen  4.6616964620084754
2  Lower Manhattan  4.8147262567494035
```



```
 What would you like to do? You can:
1.Ask a database a question
2.Get sample queries from a database
3.Get sample queries with sepecific PostgreSQL constructs
4.Add data to the PostgreSQL server
5.Get metadata about a table on the PostgreSQL server
Return the associated number with your choice!
1
What question would you like to ask?
 Note: if you would like to filter your query by a number, include a # infront of the number.
 Note: If you have a specific variable you would like to filter for, surround that word in single quotes.
What is the average price of 'tea'
        avg
0  3.382219
```

Random SQL Query Generation



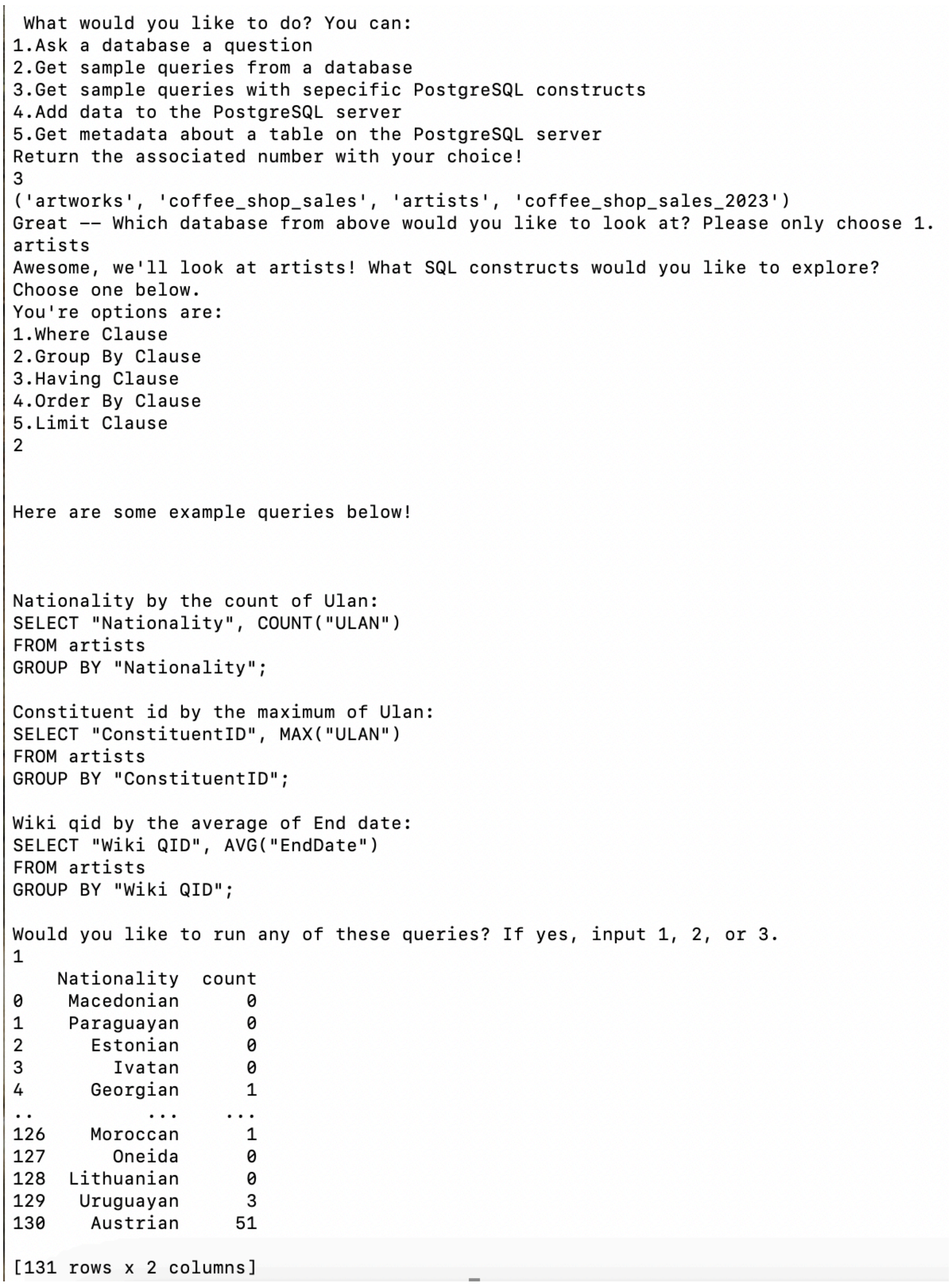
```
 What would you like to do? You can:
1.Ask a database a question
2.Get sample queries from a database
3.Get sample queries with sepecific PostgreSQL constructs
4.Add data to the PostgreSQL server
5.Get metadata about a table on the PostgreSQL server
Return the associated number with your choice!
2
('artworks', 'coffee_shop_sales', 'artists', 'coffee_shop_sales_2023')
Great -- Which database from above would you like to look at? Please only choose 1.
artworks
Here are some example queries below!
Title by the maximum of Weight (kg):
SELECT "Title", MAX("Weight (kg)")
FROM artworks
GROUP BY "Title";

Medium and Object id, filtered by Object id:
SELECT "Medium", "ObjectID"
FROM artworks
WHERE "ObjectID" = '30013';

8 Rows of On view and Artist:
SELECT "OnView", "Artist"
FROM artworks
LIMIT '8';

Would you like to run any of these queries? If yes, input 1, 2, or 3.
2
  Medium  ObjectID
0   None     30013
```

SQL Construct Queries Generation



```
 What would you like to do? You can:
1.Ask a database a question
2.Get sample queries from a database
3.Get sample queries with sepecific PostgreSQL constructs
4.Add data to the PostgreSQL server
5.Get metadata about a table on the PostgreSQL server
Return the associated number with your choice!
3
('artworks', 'coffee_shop_sales', 'artists', 'coffee_shop_sales_2023')
Great -- Which database from above would you like to look at? Please only choose 1.
artists
Awesome, we'll look at artists! What SQL constructs would you like to explore?
Choose one below.
You're options are:
1.Where Clause
2.Group By Clause
3.Having Clause
4.Order By Clause
5.Limit Clause
2


Here are some example queries below!



Nationality by the count of Ulan:
SELECT "Nationality", COUNT("ULAN")
FROM artists
GROUP BY "Nationality";

Constituent id by the maximum of Ulan:
SELECT "ConstituentID", MAX("ULAN")
FROM artists
GROUP BY "ConstituentID";

Wiki qid by the average of End date:
SELECT "Wiki QID", AVG("EndDate")
FROM artists
GROUP BY "Wiki QID";

Would you like to run any of these queries? If yes, input 1, 2, or 3.
1
    Nationality  count
0    Macedonian      0
1    Paraguayan      0
2      Estonian      0
3        Ivatan      0
4      Georgian      1
..          ...    ...
126    Moroccan      1
127      Oneida      0
128  Lithuanian      0
129   Uruguayan      3
130    Austrian     51

[131 rows x 2 columns]
```

Learning Outcomes, Challenges, and Future Work

The biggest challenge to this project was always handling the inputted natural language questions. While I believe the keyword strategy was a strong choice, the ultimate result is limited to the size of the vocabulary the administrator of the program has created for a specific data set. The vocabulary for the SQL clause detection is strong and robust, but managing the columns is unable to scale unless the user inputs exact column names. In the ideal world, the program could have been allowed to create some sort of machine learning algorithm to either associate probabilities to each word in matching it to a column name, or an algorithm to generate synonyms of a column name to build out a vocabulary automatically. Ultimately, this is the direction I would want to steer this project in going forward, as NLP is a hot button topic at the moment, and I believe my coursework in ML and deep learning could provide me with the toolkit to address this shortcoming of the current program. And while the natural language processing aspect is a bit manual, I believe that the other functionalities of the ChatDB program are very strong, which improves the overall efficacy of the project.

Conclusion

Ultimately, ChatDB is a strong program for exploring a variety of functionalities of PostgreSQL. It is able to demonstrate to a beginning user several of the core clauses of SQL syntax, allowing them to grow in their knowledge of the tool. The natural language processing aspects allow users to input specific questions to the program, allowing for curiosity to be answered with SQL queries. While the keyword matching was always a challenging problem, the robustness of matching words to SQL clauses is strong, and a major achievement of the program. Going forward, the goal will be to explore the available NLP toolkits in Python, to incorporate machine learning into the script, allowing for more accurate results and scaling beyond a few data sets.