SQL Query Generation Demo

This project demonstrates how to generate SQL queries from natural language using a large language model (LLM). The demo uses a sample SQLite database (`demo.sqlite`) and a Python script (`text_to_sql.py`) to interact with the LLM and execute the generated SQL queries.

Files

- `demo.sqlite`: An SQLite database containing dummy data for demonstration purposes.
- `prompt.txt`: A prompt template used by the LLM to generate SQL queries based on natural language input.
- `schema.txt`: A Data Definition Language (DDL) description of the data structure contained within the `demo.sqlite` database.
- `text_to_sql.py`: A Python script that demonstrates how to generate SQL queries from natural language using an LLM and execute them against the `demo.sqlite` database.

Prerequisites

Before running the script, ensure you have the following:

- Python installed on your system
- An Anthropic API key with access to the Claude model

To install the required Python packages, run the following command:

```
pip install anthropic
```

For more information on setting up the Anthropic API and obtaining an API key, refer to the Anthropic documentation.

Usage

1. Make sure you have fulfilled the prerequisites mentioned above.
2. Run the `text_to_sql.py` script using the command `python text_to_sql.py`.
3. Enter a natural language question when prompted. The script will generate an SQL query based on the provided question and execute it against the `demo.sqlite` database.
4. The script will display the results of the executed SQL query.

How it works

1. The `text_to_sql.py` script reads the `prompt.txt` template and the `schema.txt` DDL description.
2. It prompts the user to enter a natural language question.
3. The script sends the question, along with the prompt template and schema, to the LLM for processing.

4. The LLM generates an SQL query based on the provided natural language question and the database schema.
5. The script executes the generated SQL query against the `demo.sqlite` database and displays the results.

Prompting Techniques

`prompt.txt` contains a prompt template that is used to generate SQL from text. The template has the following variables:

- `{$SCHEMA}`: The database schema in SQL DDL format.
- `{$QUESTION}`: The natural language question to be translated into an SQL query.
- `{$DATABASE_TYPE}`: The type of the database (e.g., SQLite).
- `{$SIMILAR_QUERIES}`: Examples of similar questions and their corresponding SQL queries.

For more information on prompting, read the docs here.

In this prompt, we are leveraging several prompting techniques at once:

1. **Schema Specification**: By providing the database schema in SQL DDL format, we give the LLM a clear understanding of the structure and relationships within the database. This allows the LLM to generate more accurate and relevant SQL queries based on the given schema.

2. **Question Template**: The prompt includes a placeholder for the natural language question (`{$QUESTION}`). This allows us to easily inject different questions into the prompt without modifying the entire template.

3. **Database Type Specification**: The prompt includes a placeholder for the database type (`{$DATABASE_TYPE}`). This enables the LLM to generate SQL queries that are specific to the target database system, taking into account any differences in syntax or supported features.

4. **Similar Query Examples**: The prompt includes a placeholder for examples of similar questions and their corresponding SQL queries (`{$SIMILAR_QUERIES}`). By providing these examples, we give the LLM additional context and guidance on how to translate natural language questions into SQL queries. The examples serve as a reference for the LLM to understand the desired format and structure of the generated SQL.

5. **Step-by-Step Reasoning**: The prompt instructs the LLM to break down the steps and reasoning used to translate the question into an SQL query. This encourages the LLM to provide a clear and logical thought process, making the generated SQL more understandable and easier to debug or modify if needed.

6. **Iterative Refinement**: The prompt asks the LLM to provide a first draft SQL query, then review and refine it based on the original question and the desired output. This iterative process helps improve the accuracy and relevance of the generated SQL query.

7. **Structured Output**: The prompt leverages a defined XML response schema this ensures that we can parse the data in a repeatable manner.

8. **Sampling Techniques**: When generating a response from the API we ensure that the beginning and end of the sequence to be generated is fixed. To control the beginning we add a message to the

`messages` array after our user prompt, this message is the first xml tag in our response schema. This will ensure there is no pre-amble. Next we append to the `stop_sequences` array the final xml tag from the response schema. This enables a clean extraction of the contents within the xml response schema, avoiding preamble or postamble.

By combining these prompting techniques, we can guide the LLM to generate more accurate and targeted SQL queries based on natural language questions, while also providing flexibility and adaptability to different database schemas and types.