Enhancing SQL Agents with Retrieval Augmented Generation (RAG)

Luc Nguyen · Follow

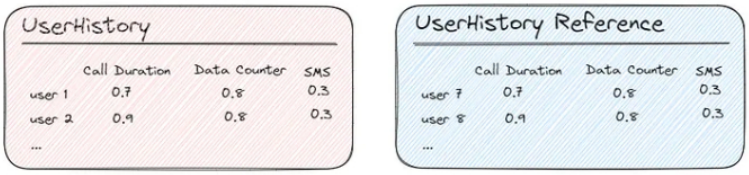
6 min read · Dec 21, 2023

In the previous blog post, we delved into the process of constructing an SQL Agent to help us answer questions by querying data in the database. In this article, let's explore how to enhance the capabilities of your SQL Agent by incorporating advanced analytics functions. Imagine that the Agent is not only capable of providing basic statistical numbers, such as the average amount of money a customer paid, but also has the ability to offer more advanced and intriguing insights. This includes tasks like identifying similarities between users or products in the database or determining the route path of users who frequently cancel their memberships. Let's discuss how to achieve these advanced functionalities.

Teradata's Advanced Analytics function

Unlike other databases, Teradata stands out by offering a plethora of advanced analytics functions, spanning from Data Cleaning and Data Exploration to Model Training, Text Analytics, and Path and Pattern Analysis functions.

The distinctive feature is that all these functions can seamlessly run ***in-database***, eliminating the need for you to set up separate environments. When you execute these functions, they are processed directly within the database, ensuring high performance.



Tables

For instance, consider two tables in the database: `UserHistory` and `UserHistoryReferences`. Using the `TD_VectorDistance` function, you can find similar users between these tables. The query syntax is as follows:

```
SELECT target_id, reference_id, distancetype, CAST(distance AS DECIMAL(36,8)) AS
FROM TD_VECTORDISTANCE (
    ON target_mobile_data_dense AS TargetTable
    ON ref_mobile_data_dense AS ReferenceTable DIMENSION
    USING
        TargetIDColumn('userid')
        TargetFeatureColumns('CallDuration','DataCounter','SMS')
```

```
        RefIDColumn('userId')
        RefFeatureColumns('CallDuration','DataCounter','SMS')
        DistanceMeasure('cosine')
        TopK(2)
) AS dt ORDER BY 3,1,2,4;
```

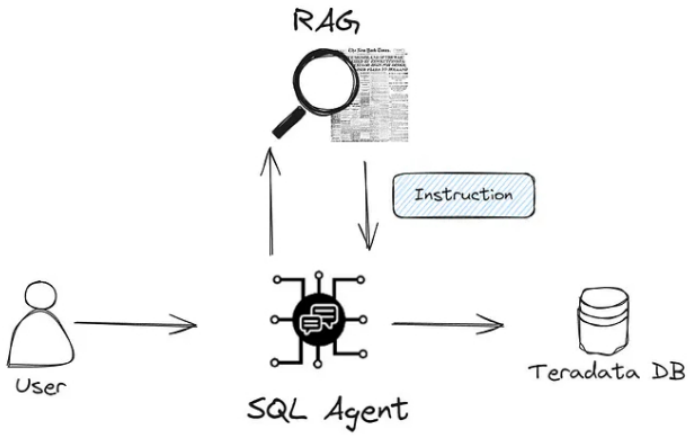
And here are result from DB:

Target_ID	Reference_ID	DistanceType	Distance
1	5	cosine	0.45486518
1	7	cosine	0.32604815
2	5	cosine	0.02608923
2	7	cosine	0.00797609
3	5	cosine	0.02415054
3	7	cosine	0.00337338
4	5	cosine	0.43822243
4	7	cosine	0.31184844

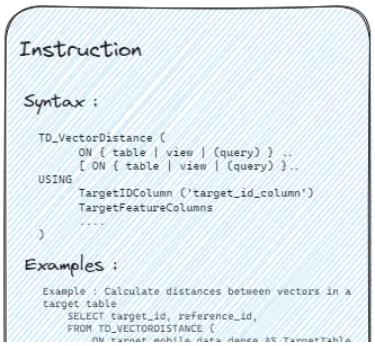
For Teradata's advanced analytics details, check the documentation at here.

Retrieval Augmented Generation (RAG)

To facilitate your agent's understanding of how to use these functions, I propose employing a technique known as Retrieval Augmented Generation (RAG).



This approach aids in locating relevant instructions based on the query. For instance, if I ask my agent to assist me in finding similar users based on the tables `UserHistory` and `UserHistoryReferences`, RAG will efficiently return the appropriate syntax and examples related to this request.



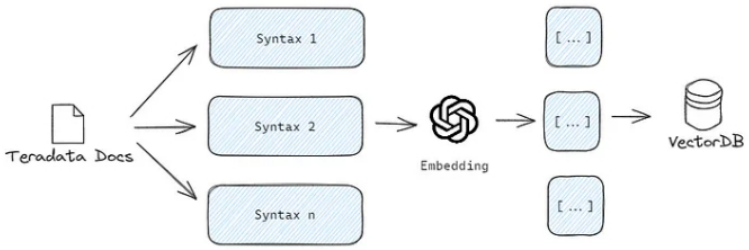
```
ON ref_mobile_data_dense AS
USING
    TargetIDColumn('id_column')
  ature2','feature3')
    DistanceMeasure('cosine')
    topk(2)
) AS dt order by 3,1,2,4;
```

Syntax Instruction

For optimal performance of the SQL Agent, Syntax Instructions should contain two essential pieces of information. First, include the ***syntax*** with explanations for each parameter. Second, and most importantly, ***provide examples***. The more examples you provide, the more accurate the SQL syntax generated by the Agent will be.

Let's Construct RAG

To create a RAG system, begin by preparing documents. Convert these documents into vectors and save them in a Vector Database, which we'll refer to as Vector DB. In this example, I'll be using a Vector DB named FAISS.




```
# Import require lib
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
```

Start by reviewing the documentation provided by Teradata. First, prepare syntax instructions that include explanations and examples.

```
syntax_1 = """
Syntax Description :
    TD_VectorDistance () ...

Example :
    TD_VectorDistance ( ... )
"""

syntax_2 = """
...
"""

syntax_3 = """
...
"""
```

Next, leverage various Open Source models from platforms like Hugging Face or OpenAI Embedding Service. In this instance, I utilized OpenAI for this task.

```
embedding_function = OpenAIEmbeddings(openai_api_key=os.getenv("OPENAI_API"))
```

Finally, with the assistance of Langchain and the FAISS Database, you can complete the process with just few line of code

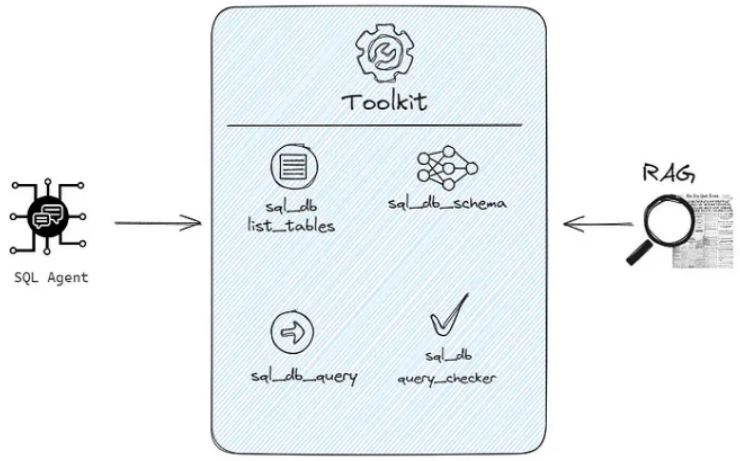
```
technical_list = [syntax_1, syntax_2, syntax_3, ..syntax_n]

db = FAISS.from_texts(technical_list, embedding_function)
```

You can easily search for relevant documents in the database using the simple code below. For instance, if you're looking for the syntax used to calculate similarity, the following code will return the exact syntax you prepared in the previous step that is relevant to your query:

```
db.similarity_search("Calculate similarity")[0]
```

Integrating RAG with SQL Agent

We have already covered how to create an SQL Agent in a previous blog post. If you are not familiar with the process, please refer to that blog. Additionally, we have discussed creating a RAG to retrieve relevant syntax instruction information. Now, let's explore how to integrate these two components seamlessly.



RAG as a tool

In this blog, I've detailed how the SQL Agent utilizes tools like `sql_db_list_tables` to interact with the database. Now, my concept is to designate RAG as another tool. This allows the SQL Agent to decide when to explore relevant documents and identify the most suitable keywords for searching when needed.

Create customize tool

To create a custom tool using Langchain, extend the `BaseTool` class provided by Langchain and customize the `_run` function as follows. It's crucial to keep the description clear to ensure that the SQL Agent understands the tool's purpose.

```
from langchain.tools import BaseTool
from typing import Optional
from langchain.callback_manager import CallbackManagerForToolRun, AsyncCallbackM

# Define retriever
retriever = db.as_retriever()

# Define customize tool
class TeradataSearchTool(BaseTool):
    name = "teradata_search_tool"
    description = "Input to this tool is a keyword such as binning or bucketing,

    def _run(
        self, query: str, run_manager: Optional[CallbackManagerForToolRun] = Non
    ) -> str:
        """Use the tool."""
        global retriever
        relevant_doc = retriever.get_relevant_documents(query)
        if len(relevant_doc) == 0 or len(query) == 0:
            return "There are no Teradata syntax examples to be used in this sce
        else:
            return relevant_doc[0].page_content

    async def _arun(
        self, query: str, run_manager: Optional[AsyncCallbackManagerForToolRun]
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("custom_search does not support async")

# Init teradata search tool
teradata_search_tool = TeradataSearchTool()
```

Create SQL Agent with customize tools

After defining the Teradata Search tool, you can create an SQL Agent using the following code. Additionally, you can include the `teradata_search_tool` created in the previous step in the `extra_tools` section.

```
# Step 4. Create Agent Executor
sql_agent = create_sql_agent(
    llm=model,
    toolkit=toolkit,
    verbose=True,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    extra_tools=[teradata_search_tool],
    prefix=prefix,
    suffix=suffix
)
```

Finally, test it

```
agent_executor.run("Identify user similarities by analyzing the 'UserHistory' ta
```

When I ask the agent to help me identify user similarities by analyzing the ‘UserHistory’ table using ‘UserHistoryReference’ as the reference table, with a focus on attributes such as CallDuration, DataCounter, and SMS, here are the results.

```
Thought: I should use the TD_VectorDistance function to calculate the distance between the t
arget pair and the reference pair from the same table.
Action: sql_db_query
Action Input: SELECT target_id, reference_id, distancetype, cast(distance as decimal(36,8))
as distance
FROM TD_VECTORDISTANCE (
    ON UserHistory AS TargetTable
    ON UserHistoryReference AS ReferenceTable DIMENSION
    USING
        TargetIDColumn('UserID')
        TargetFeatureColumns('CallDuration','DataCounter','SMS')
```

```
        RefFeatureColumns('CallDuration','DataCounter','SMS')
        DistanceMeasure('cosine')
        topk(2)
) AS dt order by 3,1,2,4;
Observation: [(1, 5, 'cosine', Decimal('0.45486518')), (1, 7, 'cosine', Decimal('0.3260481
5')), (2, 5, 'cosine', Decimal('0.02608923')), (2, 7, 'cosine', Decimal('0.00797609')), (3,
5, 'cosine', Decimal('0.02415054')), (3, 7, 'cosine', Decimal('0.00337338')), (4, 5, 'cosin
e', Decimal('0.43822243')), (4, 7, 'cosine', Decimal('0.31184844'))]
Thought: I now know the final answer
Final Answer: The user similarities can be identified by analyzing the 'UserHistory' table u
sing 'UserHistoryReference' as the reference table, focusing on attributes CallDuration, Dat
aCounter, and SMS, using the TD_VectorDistance function to calculate the distance between th
e target pair and the reference pair from the same table. The results of the query show that
users 1 and 5 have a cosine distance of 0.45486518, users 1 and 7 have a cosine distance of
0.32604815, and so on.

> Finished chain.
"The user similarities can be identified by analyzing the 'UserHistory' table using 'UserHis
toryReference' as the reference table, focusing on attributes CallDuration, DataCounter, and
SMS, using the TD_VectorDistance function to calculate the distance between the target pair
and the reference pair from the same table. The results of the query show that users 1 and 5
have a cosine distance of 0.45486518, users 1 and 7 have a cosine distance of 0.32604815, an
d so on."
```

Result from Agent

Conclusion

By combining SQL Agent with RAG, we elevate the power of the LLM model to the next level. This approach enables the creation of another RAG that empowers your agent to answer questions based on both structured data and text data. However, it's essential to acknowledge that there are still some issues related to token limits. In the next blog, I will delve into the discussion on fine-tuning the model to enable your agent to perform the same tasks without relying on RAG.

You can find full code here!

Rag | Retrieval Augmented | Llm | Teradata | Langchain

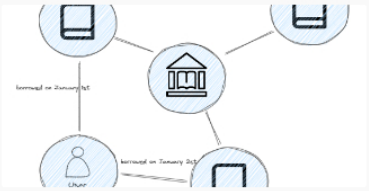
Written by Luc Nguyen

Follow

185 Followers

Data Scientist at Teradata Japan. https://www.linkedin.com/in/lucnguyenvn/

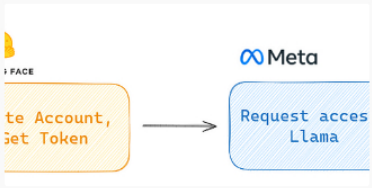
More from Luc Nguyen



Luc Nguyen

Constructing Knowledge Graphs: A Guide to Using OpenAI and Pyvis

Knowledge graphs have revolutionized the way we organize and analyze data. By visuall...



Luc Nguyen

Llama 2 Using Huggingface Part 1

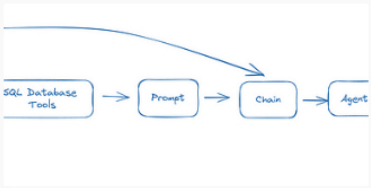
In my last blog post, I discussed the ease of using open-source LLM models like Llama...

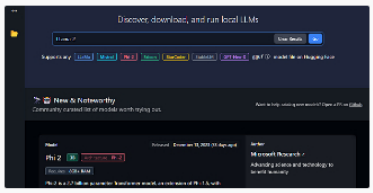
6 min read · Feb 26, 2024

233 1

5 min read · Jan 16, 2024

132





Luc Nguyen

Understanding the Magic: Deconstructing Langchain's SQL...

In my previous blog, we explored the Langchain tool and its remarkable...

5 min read · Dec 5, 2023

107 1

Luc Nguyen

Running Open Source LLM models Locally

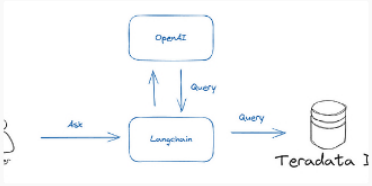
In my previous posts, I showed you how to build a chatbot connected to databases usin...

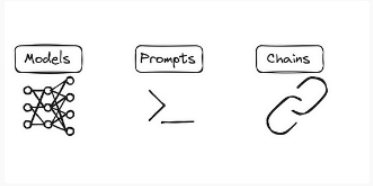
5 min read · Jan 15, 2024

121 2

See all from Luc Nguyen

Recommended from Medium





Luc Nguyen

Revolutionize Your Data Exploration: Unveiling the Power ...

In the ever-growing world of data science and machine learning, there's a game-changer o...

6 min read · Nov 27, 2023

189 1

Shivansh Kaushik

Talk to your Database using RAG and LLMs

What is RAG?

5 min read · Oct 12, 2023

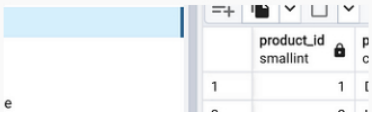
87 5

Lists

Natural Language Processing
1349 stories · 828 saves

ChatGPT prompts
47 stories · 1366 saves

Staff Picks
612 stories · 878 saves



A B Vijay Kumar

Retrieval Augmented Generation(RAG) — Chatbot for...

Implement the RAG technique using Langchain, and LlamaIndex for...

4 min read · Feb 8, 2024

121 4

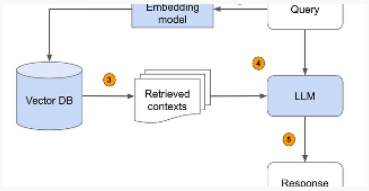
Dishen Wang in Dataherald

How to create a custom GPT from your relational database

Creating a custom GPT of real estate data using the Dataherald API

5 min read · Jan 25, 2024

205



Arun Shankar in Google Cloud - Community

Architectural Patterns for Text-to-SQL: Leveraging LLMs for...

TLDR: This article delves into the Text-to-SQL domain, demonstrating the growing reliance...

36 min read · Nov 12, 2023

459 6

Bijit Ghosh

RAG Vs VectorDB

Introduction to RAG and VectorDB

14 min read · Jan 29, 2024

151

See more recommendations