

A Smart SQL Agent: Enhancing Big Query Efficiency and Usability

Pushkar Vashishtha

Abstract: This paper presents a comprehensive methodology for developing a SQL agent leveraging LangChain to interact with Big Query. The agent facilitates seamless generation and execution of efficient SQL queries against Big Query, returning results in a readily interpretable format. We explore optimization strategies for enhancing agent performance and delve into advanced functionalities including natural language processing (NLP) powered query construction and data visualization techniques. Best practices for integrating LangChain with Big Query are also discussed, providing developers with a practical guide to building robust and performant data access solutions.

Keywords: sqlagent, bigquery, langchain, sqlalchemy, sqloptimization

1. Introduction

In today's data-driven world, efficiently querying and analyzing vast datasets is paramount. Big Query, a powerful cloud-based data warehouse, offers the scalability and performance needed for handling such data. However, interacting with BigQuery often requires specialized SQL knowledge and can be time-consuming for complex queries. This white paper introduces a novel approach to simplifying and streamlining BigQuery interactions through the development of a SQL agent powered by LangChain. This agent acts as an intelligent intermediary, allowing users to generate and execute SQL queries against BigQuery using natural language or simplified prompts. By leveraging LangChain's capabilities, the agent handles the complexities of SQL construction and execution, returning results in an easily digestible format. This empowers both technical and non-technical users to access and analyze data within BigQuery more efficiently, ultimately accelerating insights and decision-making. This paper details the methodology for constructing such an agent, exploring optimization strategies, advanced functionalities, and best practices for seamless integration with BigQuery, offering a practical roadmap for building robust and high-performing data access solutions.

2. Gen AI Agents

a) Base Level: Language Models (LLMs)

- What they do: LLMs, like the ones that power ChatGPT, are primarily focused on understanding and context within language. They can generate text, translate languages, and answer questions based on the information they've been trained on.
- Limitations: They are reactive. They respond to prompts but don't have goals, memory, or the ability to plan or interact with the world beyond text.

b) Intermediate Level: GenAI Applications

This level builds upon the foundation of LLMs, adding capabilities like:

- Conditional Data Generation: Controlling the output of the LLM more precisely. For example, specifying the length, style, or topic of the generated text.
- Creativity and Novelty: Going beyond simply regurgitating training data to generate new and original

content, like poems, code, scripts, musical pieces, email, letters, etc.

- Gen AI Applications: Utilizing these enhanced capabilities for specific applications like writing assistance, code generation, or creative content creation.

c) Advanced Level: GenAI Agents

What they do:

Gen AI agents represent a significant leap beyond LLMs. They combine the language capabilities of LLMs with reasoning, problem-solving, and agentic workflows. Crucially, they can interact with their environment, take actions, and pursue goals.

Key Differences from LLMs:

- Proactive: Agents can initiate actions without explicit human prompting. They are goal-driven and can plan steps to achieve those goals.
- Memory and Learning: Agents can retain information from past interactions and use it to inform future actions. They learn and adapt over time.
- Data Synthesis (Agentic Workflows): Agents can interact with various data sources and tools, synthesize information, and use it to achieve their goals.
- Human-Like Interactions: The ultimate aim is to create agents that can interact with humans in a natural and intuitive way, understanding complex requests and carrying out multi-step tasks. They are designed to be more autonomous and require less direct human supervision.

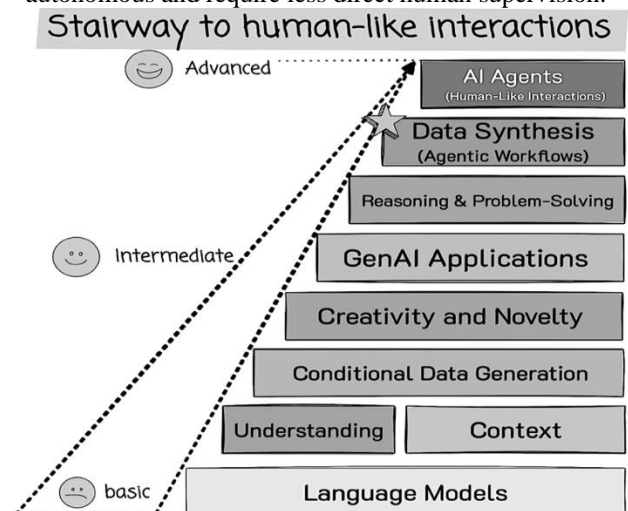


Figure 1: Gen AI Agent Evolution

3. Sql Agent using langchain and Sql Alchemy

An SQL AI agent uses Large Language Models (LLMs) to translate natural language questions into SQL queries. The agent receives a question, the LLM converts it into a SQL query, executes it against the SQL database, and then another LLM translates the result set back into a natural language answer. The SQL agent acts as the orchestrator of this process, managing the interaction between the LLMs and the database.

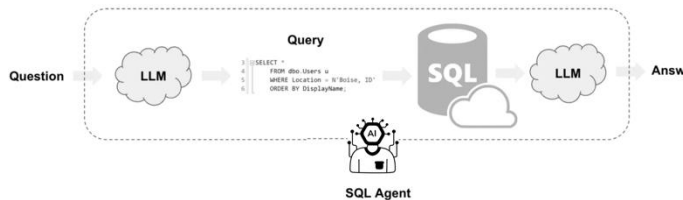


Figure 2: SQL Agent Overview

3.1 Connecting to BigQuery with SQLAlchemy

a) Creating an Agent

Once installed, you can create a connection to connect to your BigQuery project:

```
sqlalchemy_url = f'bigquery://{project_id}/{dataset_id}'
db = SQLAlchemy.from_uri(sqlalchemy_url,
                        include_tables=[table_name],
                        sample_rows_in_table_info=1)
toolkit = SQLAlchemyToolkit(db=db, llm=llm)
```

Remember to replace your-project-id with your actual BigQuery project ID, which you can find in the Google Cloud console.

b) Authentication Methods

- Service Account Key: Provide the path to your service account key JSON file:

```
sqlalchemy_url = f'bigquery://{project_id}/{dataset_id}'
db = SQLAlchemy.from_uri(sqlalchemy_url,
                        include_tables=[table_name],
                        sample_rows_in_table_info=1,
                        credentials_path='./service_account.json')
toolkit = SQLAlchemyToolkit(db=db, llm=llm)
```

- Application Default Credentials: If you've set up application default credentials, the library will automatically use them.

3.2 Creating a SQL Agent with LangChain

With the BigQuery connection established, we can now create the SQL agent using LangChain. LangChain provides a framework for building agents that can interact with various tools, including databases.

LangChain offers different types of SQL agents, each with its own approach to generating and executing queries ⁴:

- **zero-shot-react-description**: This agent type relies on a prompt to guide its behavior. It analyzes the user's query and the database schema to generate a SQL query.
- **openai-functions**: This agent type leverages OpenAI's function-calling capabilities to select and execute appropriate actions, such as generating SQL queries or retrieving table schemas.

The choice of agent type depends on your specific needs and the complexity of your application.

Basic example

```
from langchain_community.agent_toolkits.sql.toolkit import SQLDatabaseToolkit
import mock
from langchain_community.agent_toolkits.sql.base import create_sql_agent
from langchain_community.utilities import SQLDatabase
sqlalchemy_url = f'bigquery://{project_id}/{dataset_id}'

db = SQLDatabase.from_uri(sqlalchemy_url,
                        include_tables=[table_name],
                        sample_rows_in_table_info=1)
toolkit = SQLDatabaseToolkit(db=db, llm=llm)

# Create the SQL agent
sql_agent = create_sql_agent(llm=llm, toolkit=toolkit,
                            tools=toolkit.get_tools(),
                            top_k=5, verbose=True)

request_data = request.get_json()
command = request_data.get("command", "")

try:
    start_time = time.time()
    response = sql_agent.run(command)
    end_time = time.time()
    elapsed_time = end_time - start_time

    print(f"Agent's response:\n(response)")
    print(f"Time taken to process the query: {elapsed_time:.2f} seconds")
except Exception as e:
    print(f"An error occurred: {e}")

return jsonify({'response': response})
```

SQL Agent life cycle

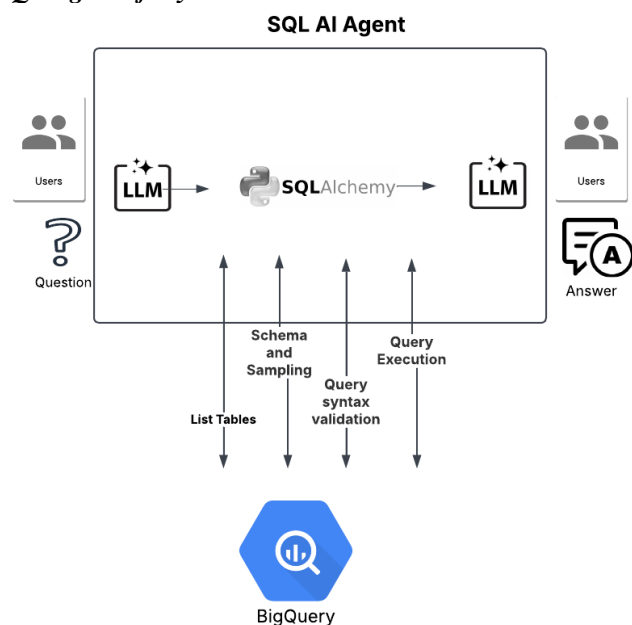


Figure 3: SQL Agent lifecycle

The SQL AI Agent lifecycle describes the process of answering a user's natural language question by leveraging large language models (LLMs), a SQL toolkit like SQLAlchemy, and a database such as BigQuery. The agent translates the question into SQL queries, executes them against the database, and then translates the results back into a natural language response. This involves several steps involving different components working together, consider: *Here's a breakdown of the SQL AI Agent Lifecycle:*

- a) User Input: The process begins with a user posing a question in natural language. This question can be about anything related to the data stored in BigQuery.

- b) Initial LLM Processing: The first LLM takes the user's question as input. It analyzes the question to understand its intent and the information requested. This stage might involve natural language processing tasks such as intent recognition, entity extraction, and preliminary query formulation.
- c) SQL Alchemy Interaction: The output from the first LLM, which might be a partially formed query or a set of instructions, is then passed to SQL Alchemy. SQL Alchemy acts as the intermediary between the LLM and the BigQuery database. It performs several crucial functions:
 - List Tables: SQL Alchemy queries BigQuery for schema information, including a list of available tables and their structure. This information is then passed back to the LLM to aid in generating a relevant SQL query.
 - Schema and Sampling: SQL Alchemy can also fetch sample data and schema details from BigQuery. This can assist the LLM in understanding the data's structure and content, allowing it to generate more effective and precise queries.
 - Query Syntax Validation: Once the LLM generates a SQL query, SQL Alchemy validates its syntax before sending it to BigQuery. This ensures that the query is structurally sound and prevents errors during execution.
 - Query Execution: If the SQL query is valid, SQL Alchemy sends it to BigQuery for execution.
- d) BigQuery Execution: BigQuery receives the validated SQL query from SQL Alchemy and executes it against the relevant dataset. The result of this execution is a data table containing the answer to the user's question in a structured format.
- e) Result Retrieval: SQL Alchemy retrieves the results of the query execution from BigQuery. This result set might contain raw data, aggregated values, or other information depending on the user's original question and the executed query.
- f) Final LLM Processing: The second LLM receives the structured data returned by BigQuery. Its task is to process these results and translate them back into a human-readable, natural language response. This stage may involve summarizing data, formatting the response, and adding context relevant to the user's initial question.
- g) Answer Output: Finally, the processed answer generated by the second LLM is presented to the user. This completes the lifecycle of answering the user's question, providing a seamless experience of interacting with the data using natural language.

Challenges with Complex Databases

When working with complex database structures or ambiguous table names, the SQL agent might face challenges in accurately understanding user queries and generating correct SQL 4. To address this, consider:

- Providing a clear database description: Include a detailed description of the database schema and table structures in the prompt to guide the agent.
- Hardcoding example questions and queries: Provide the agent with a few examples of questions and their corresponding SQL queries to help it learn the relationships between natural language and SQL.

Benefits of Using SQL Agents

SQL agents offer several advantages for interacting with databases 6:

- *Schema Awareness:* SQL agents can understand and interpret the database schema, enabling them to generate more informed and accurate queries.
- *Error Recovery:* They can handle errors gracefully by regenerating queries when encountering issues, ensuring a smoother user experience.
- *Multi-Query Handling:* SQL agents can manage complex questions that require multiple dependent queries, streamlining the data retrieval process.
- *Token Efficiency:* By focusing only on relevant tables and columns, they save tokens, making them more efficient in terms of resource usage.

4. Optimization and Improvements

Optimizing SQL Queries for Big Query

When working with large datasets in BigQuery, optimizing SQL queries for performance is essential 6. Here are some strategies to consider:

- *Use SELECT statements wisely:* Only select the columns you need to reduce the amount of data processed.
- *Filter early:* Apply WHERE clauses early in the query to limit the data scanned.
- *Leverage partitioning:* If your tables are partitioned, ensure your queries utilize this feature to minimize data scanned.
- *Avoid SELECT *:* Specify the required columns instead of retrieving all columns.
- *Use indexes:* Create indexes on frequently queried columns to speed up data retrieval.

Optimization can be achieved with above strategies and many more by providing the agent prefix context which guides the agent to create queries based on the recommendations.

Enhancing the SQL Agent with Few-Shot examples and Data Visualization

To further enhance the SQL agent, consider incorporating Few-Shot examples and data visualization.

a) Dynamic Few-Shot Example:

Improve the agent's performance by dynamically selecting relevant examples based on the user's query 13. This involves:

- *Creating a repository of examples:* Store a collection of question-query pairs.
- *Using semantic similarity:* Analyze the user's query and select the most semantically similar examples from the repository.
- *Including the selected examples in the prompt:* Provide the selected examples to the language model to guide its query generation.

b) Data Visualization

Data visualization can make the information retrieved from the database more engaging and understandable. *Visualization Techniques:* Consider using various visualization techniques to present the data, prompt the agent LLM to present data in a particular format based on the question and convert the data to relevant chart to make it more interactive.

- Column charts: Compare different categories or values.
- Bar charts: Display data with distinct categories.
- Scatter plots: Show the relationship between two variables.
- Line graphs: Illustrate trends over time.
- Pie charts: Represent proportions of a whole.

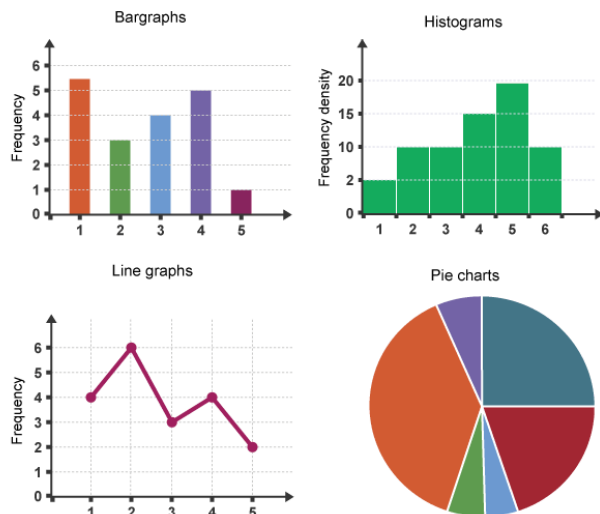


Figure 4: Charts

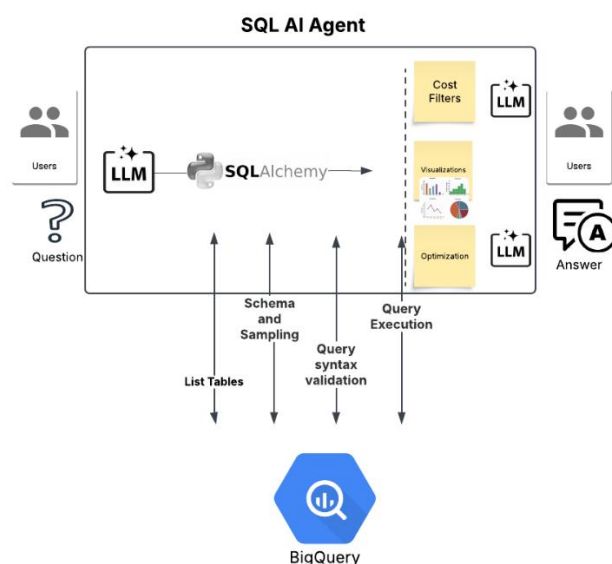


Figure 5: Enhanced Sql Agent

Enhancing Chatbots with Memory

Incorporating memory into your SQL agent can enhance its ability to handle follow-up questions and maintain context [3]. This involves:

- Storing previous interactions: Keep a record of the user's previous questions and the agent's responses.
- Referring to past context: When answering a new question, the agent can refer to the stored memory to understand the context and provide more relevant answers.

Example Scenario

User: "What were the total sales in 2023?"

Agent: "Total sales in 2023 were \$1,000,000."

User: "And in 2022?"

With memory, the agent can understand that "2022" refers to the sales in the previous year and provide the correct answer without needing the user to repeat the entire context.

Caching with data portal metadata APIs

To enhance the performance of our SQL agent tool, we are investigating methods to optimize metadata retrieval. Currently, the agent fetches metadata directly for each query. Our proposed solution involves leveraging internal APIs, such as the Data portal assets API or custom table metadata APIs, to retrieve this information. By caching the retrieved metadata, we can drastically reduce the number of API calls and the associated latency. This caching strategy aims to significantly improve the overall performance and responsiveness of the SQL agent, especially for frequently accessed tables. We expect this optimization to minimize query execution time and enhance the user experience.

5. Conclusion


By combining the capabilities of LangChain, BigQuery, and SQL Alchemy, you can create a powerful SQL agent that can effectively interact with your data and provide valuable insights. Incorporating advanced techniques like NLP and data visualization, along with following best practices, can further enhance the agent's performance and user experience. This opens up a wide range of applications for SQL agents in various domains, from data analysis and reporting to customer service and decision-making.

This article provided a comprehensive guide to building and enhancing a SQL agent with LangChain and BigQuery. We started by establishing a connection to BigQuery using SQL Alchemy, exploring different authentication methods and connector options. Then, we delved into creating the SQL agent with LangChain, discussing various agent types and addressing challenges with complex databases. We highlighted the benefits of using SQL agents, such as schema awareness and error recovery, and explored techniques for generating, executing, and validating SQL queries. We also discussed strategies for optimizing queries for BigQuery and presenting the output in a user-friendly format. Finally, we explored advanced techniques like NLP and data visualization to further enhance the agent's capabilities and discussed best practices for using LangChain with BigQuery.

References

- [1] Rangchekar, M. (n.d.). *Getting started: Building your first SQL database agent with LangChain*. Medium. Retrieved February 12, 2025, from <https://medium.com/@mandarangchekar7/getting-started-building-your-first-sql-database-agent-with-langchain-bc8b45a9ba70>
- [2] Agents | LangChain. (n.d.). Retrieved February 12, 2025, from https://python.langchain.com/v0.1/docs/use_cases/sql/agents/
- [3] *Langchain knowledge SQL agent BigQuery cat AI*. (n.d.). Restack. Retrieved February 12, 2025, from <https://www.restack.io/docs/langchain-knowledge-sql-agent-bigquery-cat-ai>
- [4] *SQL agent for Google Big Query · Discussion #17760 · langchain-ai/langchain*. (n.d.). GitHub. Retrieved February 12, 2025, from <https://github.com/langchain-ai/langchain/discussions/17760>

February 12, 2025, from <https://github.com/langchain-ai/langchain/discussions/17760>

- [5] *Using LangChain and GPT to chat with your BigQuery data.* (n.d.). DataScienceEngineer. Retrieved February 12, 2025, from <https://www.datascienceengineer.com/blog/post-chat-with-bigquery>
- [6] *SQL -  LangChain.* (n.d.). Retrieved February 12, 2025, from https://python.langchain.com/v0.1/docs/use_cases/sql/
- [7] *Best practices for SQL query optimizations.* (n.d.). GeeksforGeeks. Retrieved February 12, 2025, from <https://www.geeksforgeeks.org/best-practices-for-sql-query-optimizations/>