**Documenting the SQL Querying System Using CrewAI and LangChain**

**-Shivam Tarte**

**Overview**

This document provides a comprehensive explanation of the SQL querying system I implemented using **CrewAI**, **LangChain**, and **MySQL**. The system is designed to facilitate automated SQL query generation, validation, and execution using an **LLM (Large Language Model)**.

The implementation connects to a **sample database** called world and enables intelligent SQL operations by leveraging **LangChain tools** and **CrewAI agents**. I have also included code execution logs and outputs for reference.

**Technology Stack Used**

The project is built using the following technologies:

* **Python** – The main programming language for implementation.
* **MySQL** – A relational database used for querying and fetching data.
* **SQLAlchemy** – Used to create a database connection engine.
* **PyMySQL** – A MySQL client for Python to facilitate connections.
* **LangChain** – Provides useful tools for working with SQL databases.
* **CrewAI** – A framework to structure AI-driven agents and workflows.
* **Ollama/Mistral LLM** – Used as the Large Language Model for generating SQL queries.

**Database Details**

I used the **sample MySQL database** named world, which contains:

* city table
* country table
* countrylanguage table

This database allows testing SQL queries on real-world structured data.

**Project Implementation**

**1. MySQL Connection Setup**

To establish a connection with MySQL, I used SQLAlchemy and PyMySQL:

self.engine = create\_engine(f"mysql+pymysql://{db\_user}:{db\_password}@{db\_host}/{db\_name}")

The credentials used:

* DB\_USER = "root"
* DB\_PASSWORD = "Arcsaber0001"
* DB\_HOST = "localhost"
* DB\_NAME = "world"

The SQLDatabase.from\_uri() function initializes a LangChain-compatible database object.

**2. CrewAI Implementation**

CrewAI enables the structuring of AI-driven workflows. I created an **agent** responsible for handling SQL queries:

self.sql\_agent = Agent(

name="SQL Writer",

role="Database Query Expert",

goal="Generate optimized and accurate SQL queries based on user requirements.",

backstory="""

You are an experienced database engineer who is an expert at writing and optimizing SQL queries.

Use `list\_tables` to find available tables.

Use `tables\_schema` to understand the table structures.

Use `check\_sql` to verify query correctness.

Use `execute\_sql` to run queries safely.

""",

llm=SQLAgent.llm,

tools=[SQLAgent.list\_tables, SQLAgent.tables\_schema, SQLAgent.check\_sql, SQLAgent.execute\_sql],

allow\_delegation=False,

verbose=True,

)

This agent has access to four main tools:

1. list\_tables – Lists all tables in the database.
2. tables\_schema – Retrieves schema information for specific tables.
3. check\_sql – Validates SQL queries before execution.
4. execute\_sql – Executes SQL queries.

I also defined a **CrewAI Task**:

self.extract\_data = Task(

description="Extract data that is required for the query {query}.",

expected\_output="Database result for the query",

agent=self.sql\_agent,

)

Finally, I created a **Crew** to handle sequential task execution:

self.crew = Crew(

agents=[self.sql\_agent],

tasks=[self.extract\_data],

process="sequential",

verbose=True,

memory=False,

output\_log\_file="crew.log",

)

The crew.log file logs all query execution and AI processing.

**3. Query Execution Flow**

To execute user queries, I defined the run\_query method:

def run\_query(self, user\_query: str):

"""Runs a user-provided SQL query through the CrewAI system."""

inputs = {"query": user\_query}

result = self.crew.kickoff(inputs=inputs)

return result

This function sends the query request to CrewAI and returns the output.

**Example Query Execution**

To test the system, I executed a query to **find the country with the largest surface area**:

sql\_agent\_instance = SQLAgent(DB\_USER, DB\_PASSWORD, DB\_HOST, DB\_NAME)

try:

result = sql\_agent\_instance.run\_query("country having largest surface area")

print("Query Result:\n", result)

except Exception as e:

print(f"An error occurred: {e}")

**Expected Output**

The output should be an SQL query that retrieves the country with the maximum surface area:

SELECT Name, SurfaceArea FROM country ORDER BY SurfaceArea DESC LIMIT 1;

And the result (example output):

+--------------+-------------+

| Name | SurfaceArea |

+--------------+-------------+

| Russia | 17098242.00 |

+--------------+-------------+

**Logging Input and Output**

To keep track of inputs and results, I have enabled logging in crew.log. Example logs:

[INFO] Query Received: country having largest surface area

[INFO] Generated SQL: SELECT Name, SurfaceArea FROM country ORDER BY SurfaceArea DESC LIMIT 1;

[INFO] Query Execution Successful

[INFO] Output: [(Russia, 17098242.00)]

**Conclusion**

This project successfully implements a **CrewAI-based SQL Querying System** that allows:

* AI-driven SQL query generation
* Query validation before execution
* Secure and efficient query execution

The system is highly extensible and can be integrated with other databases or enhanced with more advanced AI models for better SQL generation and optimization.