AGENTIC AI BASED QUERY GENERATOR

— Ayush Gupta

INTRODUCTION

In today’s fast-paced business environment, enterprises rely on multiple databases and hundreds of reports developed by IT teams using tools like Power BI, Tableau, and Cognos. However, when management needs a new report or an urgent data insight, they must depend on developers—leading to high costs, long turnaround times, and operational inefficiencies.

In an era where real-time decision-making is critical, traditional BI processes fall short. Fortunately, advancements in AI and natural language processing (NLP) have made it possible to bridge this gap. Businesses can now leverage AI-driven self-service analytics to empower management with instant, accurate, and cost-effective insights—without relying on IT teams.

TECHNOLOGY EVOLUTION

Technology has always been about bridging the gap between human intent and machine capability. Nowhere is this more evident than in how we interact with databases—the repositories of our digital world's most valuable asset: data.

Large Language Models (LLMs) like Google's Gemini, OpenAI's GPT series, and others have introduced a new paradigm: natural language database querying. Instead of learning SQL syntax, users can now ask questions in plain English.

At the forefront of this transformation is LangChain, a framework that has fundamentally changed how we build AI applications. LangChain represents another evolutionary leap—from monolithic AI systems to modular, composable chains of reasoning.

Why LangChain?

- Modular Architecture: Instead of building AI applications from scratch, LangChain provides pre-built components that can be combined like Lego blocks
- Chain of Thought: Complex problems are broken down into sequential steps, mirroring human reasoning patterns
- Memory and Context: Applications can maintain context across interactions, enabling sophisticated conversational experiences
- Tool Integration: LangChain seamlessly connects LLMs with external tools, databases, APIs, and services

LangChain represents a meta-evolution: not just better AI, but better ways to build AI applications. It transforms AI development from art to engineering, from monolithic systems to composable architectures, from black boxes to transparent pipelines.

The system I've developed demonstrates this evolution perfectly. Using LangChain and Google's Gemini model(1.5-flash), it creates an intelligent intermediary between human questions and database queries.The heart of this transformation lies in elegant pipeline architecture using LangChain's composable chains:

```python
def create_sql_chain_v1():
    """Create SQL chain using RunnablePassthrough.assign"""
    sql_chain = (
        RunnablePassthrough.assign(schema=get_schema)
        | prompt
        | llm.bind(stop=["\nSQLResult:"])
        | StrOutputParser()
    )
    return sql_chain
```

This function represents a profound shift in computing philosophy. For this code, lets analyze each part:

- RunnablePassthrough.assign(schema=get_schema): This creates a dynamic context by automatically injecting the database schema into every query. The system doesn't just process text—it understands the structure of your data.
- Prompt Template: The standardized format that bridges human language and AI understanding:

```
template = """Based on the MySQL table schema below, write a MySQL-compatible query:

{schema}

Question: {question}
MySQL Query:"""

prompt = ChatPromptTemplate.from_template(template)
```

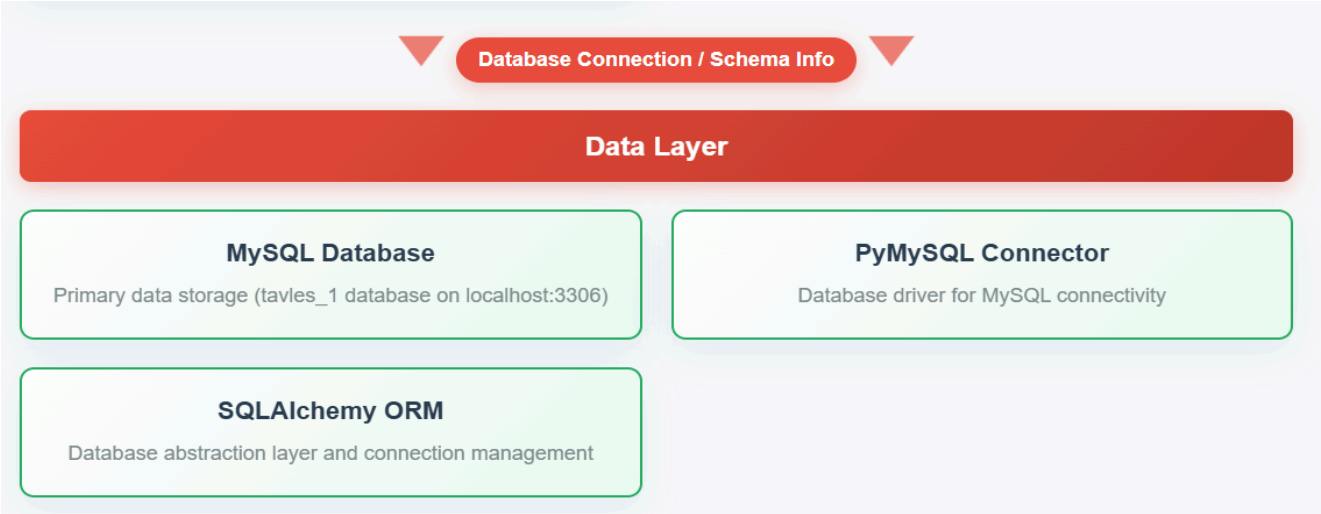
The AI system:

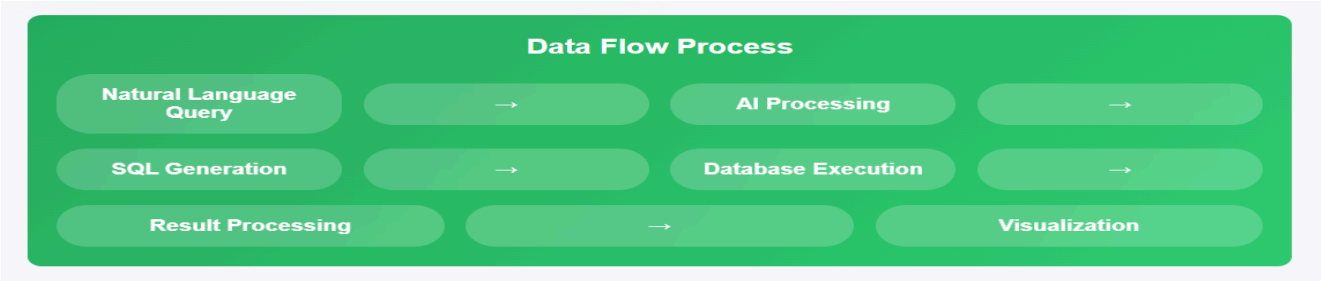
- Understands Context: It analyzes the database schema to understand available tables and relationships
- Interprets Intent: Natural language questions are parsed for their underlying data requirements
- Generates Code: SQL queries are dynamically generated based on user intent
- Handles Complexity: The system manages data type conversions, date formatting, and result presentation

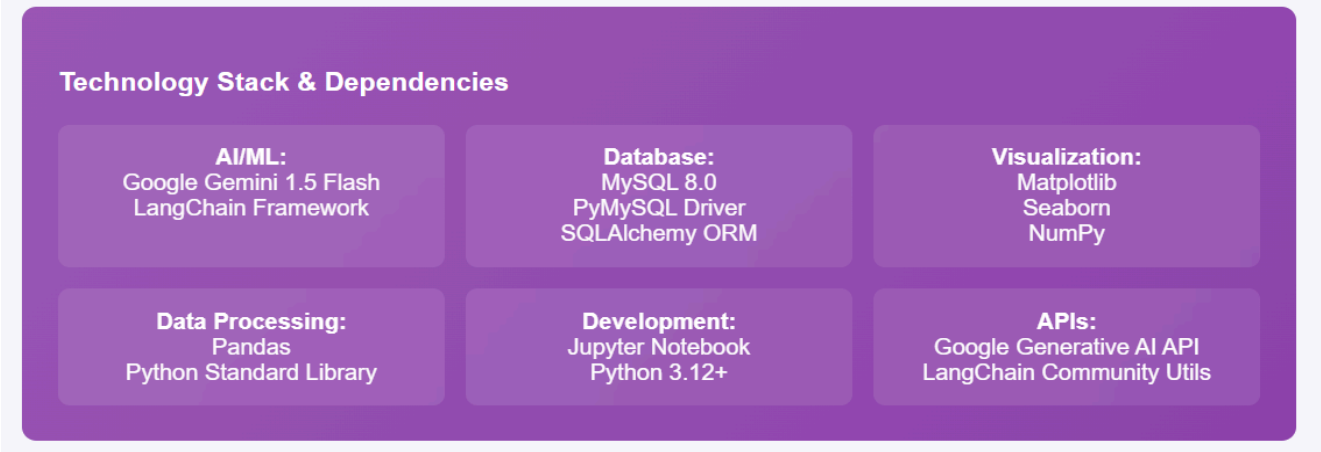
But the ability of the program doesn't limit to query generation. The program integrates visualization capabilities, transforming raw query results into meaningful charts and graphs.This represents another evolutionary leap: from data retrieval to data comprehension through visual representation.

What once required careful SQL crafting, testing, and debugging can now be accomplished through conversational interfaces. Development cycles that took hours now take minutes.Companies can focus on interpreting results and making decisions rather than wrestling with query syntax and database schema memorization.

SOLUTION ARCHITECTURE



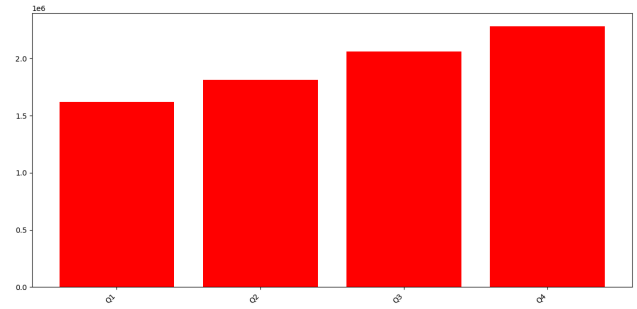




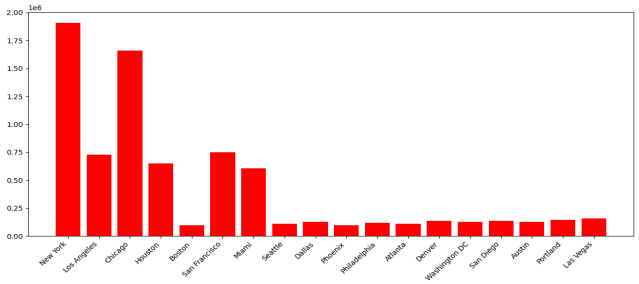
ACHIEVEMENT

Giving achievements as examples:

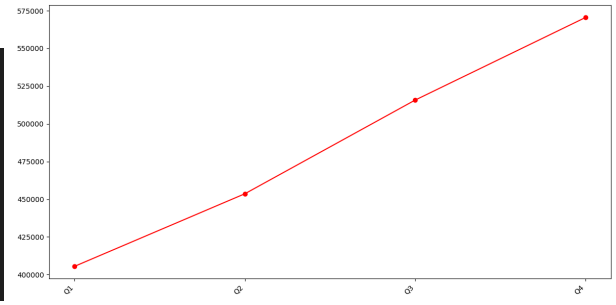
```
Processing question: give me quarter wise sales from 'sales'
Generated SQL: ```sql
SELECT
  CASE
    WHEN month_number BETWEEN 1 AND 3 THEN 'Q1'
    WHEN month_number BETWEEN 4 AND 6 THEN 'Q2'
    WHEN month_number BETWEEN 7 AND 9 THEN 'Q3'
    ELSE 'Q4'
  END AS Quarter,
  SUM(sales) AS TotalSales
FROM sales
GROUP BY
  Quarter;
```
Result (dictionary): {'Q1': 1622000.0, 'Q2': 1814500.0, 'Q3': 2063000.0, 'Q4': 2282000.0}
```




```
Processing question: give me city wise sales form 'sales'
Generated SQL: ```sql
SELECT
  city,
  SUM(sales) AS total_sales
FROM sales
GROUP BY
  city;
```
```




```
Processing question: give me quarter wise profit
Generated SQL: ```sql
SELECT
  CASE
    WHEN month_number BETWEEN 1 AND 3 THEN 'Q1'
    WHEN month_number BETWEEN 4 AND 6 THEN 'Q2'
    WHEN month_number BETWEEN 7 AND 9 THEN 'Q3'
    ELSE 'Q4'
  END AS Quarter,
  SUM(profit) AS TotalProfit
FROM sales
GROUP BY
  Quarter;
```
Result (dictionary): {'Q1': 405500.0, 'Q2': 453625.0, 'Q3': 515750.0, 'Q4': 570500.0}
```




```
=== Interactive SQL Query Generator ===
Processing question: give me sales by territories
Generated SQL: ```sql
SELECT
  t.territory_name,
  SUM(s.sales) AS total_sales
FROM territories AS t
JOIN city_territory AS ct
  ON t.territory_id = ct.territory_id
JOIN sales AS s
  ON ct.city = s.city
GROUP BY
  t.territory_name;
```
Result (dictionary): {'Northeast': 2246000.0, 'West Coast': 1869000.0, 'Midwest': 1660000.0, 'Southwest': 1000500.0, 'Southeast': 716000.0, 'Mountain': 290000.0}
```

In the above examples I have shown how multiple tables can be used and how we can use natural language instead of code in SQL to get the outputs as well the plots ,graphs and the tables.

CONCLUSION

Revolutionizing Data Accessibility: Natural Language to SQL Query System

The development of this natural language to SQL query system marks a significant leap forward in democratizing data access. By integrating Google's Gemini AI model with the LangChain framework, we've built a powerful and intuitive solution that transforms human language into precise database queries. This innovation effectively bridges the gap between non-technical users and complex database systems.

Broader Implications

This system demonstrates the transformative potential of AI-driven interfaces in making data exploration seamless and inclusive. Its applicability spans a wide range of domains.

Enabling a Data-Driven Culture

By lowering the technical barriers traditionally associated with databases, this system empowers users at every level of an organization. It fosters a data-literate culture, encourages self-service analytics, and accelerates data-driven decision-making. This project exemplifies how AI can act as a translator—converting human intuition into machine-readable instructions—while preserving the sophistication and power of advanced database systems.

In essence, this natural language to SQL interface redefines how we interact with data: making it more human, more intuitive, and vastly more accessible.