FINAL CAPSTONE PROJECT REPORT

Project Title: Enterprise-Grade Agentic Text-to-SQL System

Submitted By: Anubhav Agrawal

Date: 16th Jan, 2026

Trainer: TamalKant Mukherjee

TABLE OF CONTENTS

---

1. EXECUTIVE SUMMARY

This project delivers an "Agentic Text-to-SQL System" designed to democratize data access within an enterprise. By leveraging Large Language Models (LLMs) via Groq (Llama 3.3) and a stateful orchestration engine (LangGraph), the system allows non-technical users to query complex SQL databases using natural language.

Key differentiators include a "Zero-Trust" PII filtering mechanism that protects sensitive customer data, a self-correcting agentic loop that fixes SQL errors automatically, and a real-time visualization layer built with Streamlit.

---

2. PROBLEM DEFINITION

The Challenge: In modern enterprises, valuable data is locked in relational databases (SQL). Business stakeholders (Sales Managers, Executives) typically lack the SQL skills required to extract insights. This creates a bottleneck where every data request must go

through a technical data analyst, causing delays in decision-making.
Thus, this will help millions of people to use data in their daily life with Simple natural language.

Existing Solutions & Limitations:

- **Dashboards:** Static and cannot answer ad-hoc questions (e.g., "Why did sales drop yesterday?").
- **Basic Chatbots:** often "hallucinate" (make up data) or fail to understand complex schema relationships or wrong inputs.

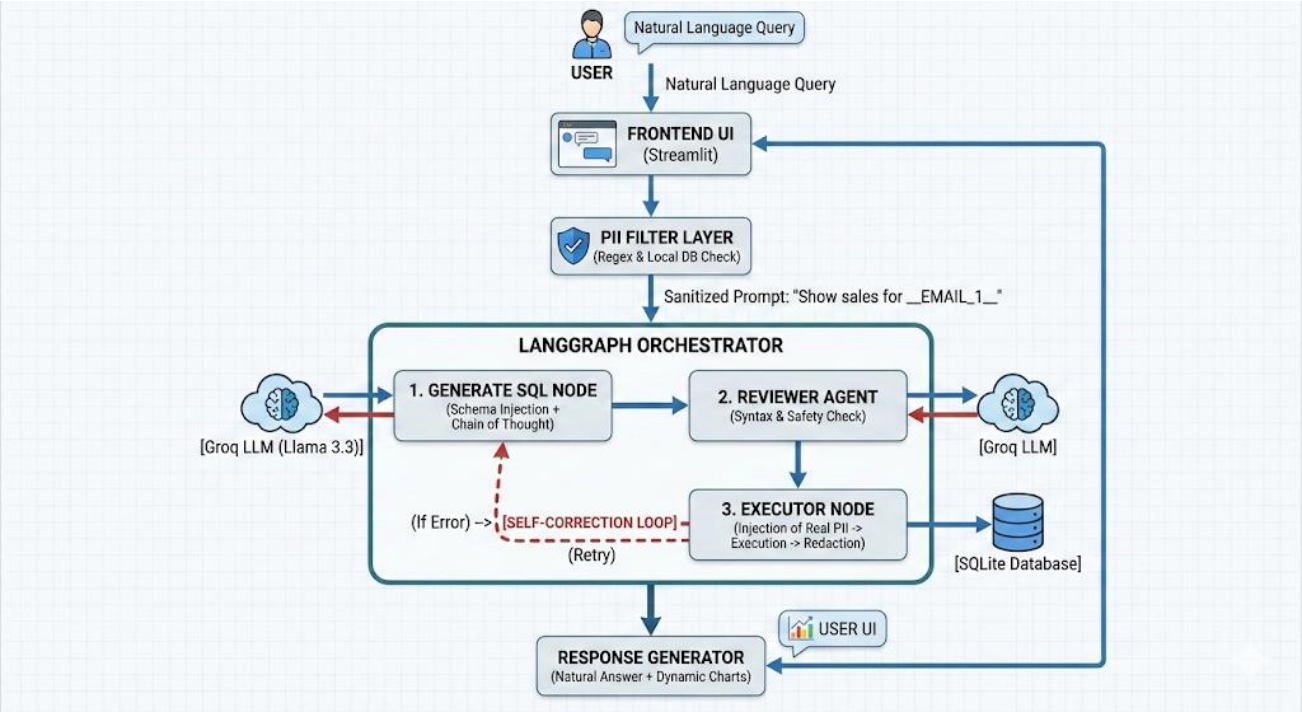
The Solution: An intelligent, secure, and interactive interface that translates natural English questions into optimized SQL queries, executes them safely, and visualizes the results, all while scrubbing sensitive PII.

---

3. SYSTEM ARCHITECTURE & DESIGN

3.1 High-Level Architecture Diagram

The system follows a "Human-in-the-Loop" Agentic architecture.



3.2 Component Breakdown

1. **Orchestrator (LangGraph):** Manages the state machine. It decides if the agent should move forward to execution or loop back to fix an error based on the should_retry conditional edge.
2. **Safety Layer (PII Filtering):** Implements a "Placeholder Swap" technique. Real emails (e.g., alice@example.com) are swapped with tokens (__EMAIL_1__) before reaching the LLM, and swapped back only during execution.
3. **Visualizer (Streamlit**): Automatically detects if the result is tabular or numeric and renders interactive Bar Charts and Dataframes.

---

4. TECHNOLOGY STACK

- **Language:** Python 3.10+
- **Frontend:** Streamlit
- **LLM Provider:** Groq (Model: llama-3.3-70b-versatile)
- **Orchestration:** LangGraph (StateGraph, Nodes, Edges)
- **Database:** SQLite (Local relational DB with multiple tables)
- **Data Handling:** Pandas (Data manipulation), Faker (Synthetic data generation)
- **Visualization:** Graphviz (Schema diagrams), Streamlit Charts

---

5. PROJECT PROGRESS REPORTS (Summary)

- **Phase 1: Database Design:** Created a robust schema with 4 tables (sales, customers, products, regions) and populated it with 200+ rows of realistic dummy data using Faker.
- **Phase 2: Baseline Agent:** Implemented a basic "Prompt-to-SQL" chain using LangChain and Groq.
- **Phase 3: Agentic Loops:** Migrated to LangGraph to handle errors. Added the check_sql_node for syntax verification and the should_retry loop for runtime error handling.
- **Phase 4: Advanced Security:** Implemented the "Zero-Trust" PII filter that redacts emails based on database presence.
- **Phase 5: UI & Visualization:** Built the Streamlit interface with the "Database Explorer" sidebar and dynamic charting capabilities.

---

6. RUNBOOK / DEPLOYMENT GUIDE

Prerequisites

- Python 3.10 or higher installed.
- A valid Groq API Key.

Step-by-Step Setup

1. Clone/Unzip Project: Extract the submitted final_project.zip folder.
2. Install Dependencies: Open a terminal in the project folder and run:

   pip install -r requirements.txt
3. Configure Environment: Open the backend.py file and ensure the API Key is set:

   GROQ_API_KEY = (Used in the project)

4. Initialize Database: Generate the fresh dataset:

   python database.py

   Output: Database updated with Customers successfully.
5. Launch Application: Run the frontend server:

   Python -m streamlit run main.py
6. Access UI: Open your browser to http://localhost:8501.

---

7. KEY LEARNINGS AND INSIGHTS

Challenges Faced

1. **LLM Hallucinations**: The model initially tried to query columns like "dance_date" which didn't exist.
   - ***Solution:*** We implemented "Schema Injection" in the system prompt and added a strict rule: "If column missing, return IMPOSSIBLE".
2. **Chatty LLM Responses:** The validation step failed because the LLM would write essays instead of just code.
   - ***Solution:*** Used Regex parsing to extract content solely between ```sql blocks.

3. **Data Privacy**: How to query specific customers without leaking their data to the cloud?
    - ***Solution:*** The "Placeholder Swap" technique (sending __EMAIL_1__ to the cloud, executing with real data locally).

Takeaways

- **Agentic > Linear:** A self-correcting loop is infinitely more reliable than a single-shot prompt.
- **Context should always be clear:** The quality of SQL generation is directly dependent on how clearly the Schema is presented in the prompt.

---

8. DEMO SCREENSHOTS

(Please insert your screenshots in the spaces below)

Figure 1: The Chat Interface with Dynamic Visualization

[INSERT SCREENSHOT OF A QUERY LIKE "Top 3 Products" SHOWING THE BAR CHART]

Figure 2: The PII Security Alert Mechanism

[INSERT SCREENSHOT OF THE " Security Alert" MESSAGE WHEN ASKING FOR AN EMAIL]

Figure 3: The Self-Correction/Schema Diagram

[INSERT SCREENSHOT OF THE SIDEBAR SCHEMA DIAGRAM WITH ARROWS]

---

9. FINAL PRESENTATION DECK (Outline)

1. **Title Slide: Enterprise Text-to-SQL Agent.**
2. **The Problem: "Data is hard to access."**
3. **The Solution: "Chat with your Database."**
4. **Live Demo:**
    - ***Query 1 (Simple):*** **"Show me total sales."**
    - ***Query 2 (Complex):*** **"Who are the top 3 customers in Europe?"**
    - ***Query 3 (Safety):*** **"Is admin@gmail.com in the database?" (Show Redaction).**

5. **Architecture: Show the LangGraph flow (Generate -> Check -> Execute).**
6. **Future Scope: Adding Voice Input and Multi-Database support.**
7. **Q&A.**

[End of Report]