

# Assignment 3

## AI-Powered SQL Query & Visualization System

*A Multi-Agent LangGraph + Streamlit Analytical Platform*

### 1. Introduction

Modern data analysis workflows increasingly rely on automation, natural-language interfaces, and AI-driven interpretation layers. This project presents a **fully automated AI-powered analytical system** that converts **natural-language questions** into SQL queries, retrieves data from PostgreSQL, generates visualization code, validates the code, executes it safely, and displays the output through a Streamlit-based user interface.

The system is powered by:

- **LangChain + LangGraph** for multi-agent orchestration
- **PostgreSQL** for structured data storage
- **Polars** for high-performance CSV ingestion
- **Streamlit** for an interactive user interface
- **OpenAI GPT-4o-mini** for semantic reasoning, SQL generation, and visualization logic

This report explains the full architecture, agent design, database setup, implementation steps, key features, and how the system meets all assignment requirements.

### 2. System Overview

The system implements a **five-agent architecture** where each agent specializes in a single computational task:

1. **Agent 1 – SQL Generator**
2. **Agent 2 – Visualization Code Generator**
3. **Agent 3 – Code Validator**
4. **Agent 4 – Secure Code Execution (Python REPL)**
5. **Agent 5 – Streamlit UI Controller**

These agents communicate using **LangGraph**, ensuring deterministic control flow, error recovery, and safe execution.

### 3. Database Design (3NF)

The PostgreSQL database is stored under the schema `robot_vacuum` and follows **Third Normal Form (3NF)**.

Key characteristics:

- No repeating groups or redundant data
- Clear entity separation (orders, products, customers, reviews, warehouses, shipments, manufacturers)
- Referential integrity maintained through foreign keys

#### 3.1 CSV Loading with Polars

A Jupyter notebook (`create_table_and_load_CSV.ipynb`) handles:

- Table creation
- Data cleaning
- Polars-based CSV ingestion (significantly faster than pandas)
- Batch insertion into PostgreSQL

This ensures a stable and performant analytical backend.

### 4. Multi-Agent Architecture

The central objective of this project is to use **multiple AI agents** working together.

Below is an explanation of each agent.

#### 4.1 Agent 1 — SQL Generator

- Input: Natural-language user question
- Output: **Valid SQL query + pandas DataFrame**
- Reads the **live PostgreSQL schema** dynamically
- Ensures:
  - No hallucinated columns
  - Proper JOINs
  - Syntax correctness
- SQL is executed using Polars via SQLAlchemy for speed
- If SQL is invalid, the agent retrieves with corrective instructions

 File: `src/agent_sql_generator.py`

## 4.2 Agent 2 — Visualization Code Generator

- Input: DataFrame + user request
- Output: **Pure executable Python visualization code**
- No placeholders or undefined variables
- Detects:
  - Date columns → line charts
  - Category + numeric → bar charts
  - Distribution/percentage → pie chart
  - Two numeric columns → scatter plot
  - One numeric column → histogram
- Builds code compatible with **Python REPL execution**

 File: `src/agent_code_generation.py`

## 4.3 Agent 3 — Visualization Code Validator

- Ensures code is:
  - Syntactically correct
  - Only using valid DataFrame columns
  - Free from unsafe patterns

 File: `src/agent_code_validator.py`

## 4.4 Agent 4 — Secure Code Runner

- Executes code using a **sandboxed Python REPL**
- Blocks unsafe imports such as:
  - `os, subprocess, sys, shutil`
- Prevents:
  - File system writes
  - Shell execution
  - Arbitrary code execution
- Returns PNG images extracted from Matplotlib/Plotly

 File: `src/agent_code_runner.py`

## 4.5 Agent 5 — Streamlit UI Application

This is the user-facing layer.

Features:

- ChatGPT-style conversational UI
- Shows each pipeline step:
  - SQL generated
  - DataFrame preview
  - Visualization code
  - Validation results
  - Final rendered chart
- Auto-detects whether the user wants:
  - A **table/text answer**, or
  - A **chart visualization**

 File: `src/streamlit_sql_agent.py`

## 5. Environment & Configuration

A `.env` file is used for secure configuration.

### Environment Variables

```
PG_HOST=localhost
PG_PORT=5433
PG_USER=postgres
PG_PASSWORD=admin
PG_DB=robot_vacuum
OPENAI_API_KEY=your_api_key_here
```

This allows the system to:

- Connect to PostgreSQL
- Authenticate with OpenAI
- Maintain portability

## 6. Running the System

### 6.1 Install Dependencies

```
pip install -r src/requirements.txt
```

### 6.2 Start Streamlit App

```
streamlit run src/streamlit_sql_agent.py
```

### 6.3 Usage Flow

1. Enter a natural-language question

Example:

“Plot monthly revenue trends over time”

2. System automatically:
  - o Generates SQL
  - o Retrieves DataFrame
  - o Determines if output should be table or chart
  - o Generates visualization code
  - o Validates safely
  - o Executes securely
  - o Presents final PNG output

## 7. Example Outputs

### 7.1 Table Examples

- “Which warehouses are below restock threshold?”
- “Which ZIP code has the highest delayed deliveries?”

### 7.2 Chart Examples

- Monthly revenue trends
- Delivery status distribution (pie chart)
- Average shipping cost by carrier (bar chart)
- Review rating by manufacturer

## 8. Security Measures

To ensure robustness and avoid malicious execution:

- Strict REPL sandbox
- No access to:
  - o Filesystem
  - o OS commands
  - o Network
- Validation prevents:
  - o Undefined references
  - o Unsafe imports
  - o Bad syntax
- Only generates visualization-focused Python code

## 9. Assignment Requirements (All Completed)

Requirement	Status
PostgreSQL tables in 3NF	✓
CSV loaded via Polars	✓
Five-agent LangGraph architecture	✓
SQL → Viz → Validation → Execution pipeline	✓
Modular code (multiple files)	✓
Streamlit UI	✓
Notebook for table creation & CSV load	✓
requirements.txt	✓
README + Report	✓

## 10. Conclusion

This project demonstrates a high-level integration of:

- AI reasoning
- Database querying
- Data visualization
- Validation
- Secure execution
- UI interactivity

The system successfully transforms natural-language questions into actionable analytical insights, fulfilling all assignment requirements and representing a production-grade AI analytics pipeline.