AI Agents Assignment Report

Build-Your-Own SQL Database Agent (ReAct Framework)

Name: Mohit Karande

UID: 2022600023

Institute: Sardar Patel Institute of Technology

Date: November 2025

ABSTRACT

This report presents the design and implementation of a ReAct-based SQL Database Agent developed entirely from scratch.

The project demonstrates the integration of reasoning and action through a loop where a Large Language Model (LLM) - Google Gemini 2.5 Flash - interprets natural language queries, reasons through available tools, performs database actions, and produces grounded answers.

The core objective of this assignment was to design and implement an agent framework without relying on any existing libraries such as LangChain or AutoGen. This implementation showcases how an LLM can be combined with a custom-built tool registry, SQL validation system, and reasoning loop to interact safely and intelligently with structured data.

1. INTRODUCTION

Agent-based systems are increasingly used to automate reasoning and interaction tasks. In this project, an autonomous SQL Database Agent was implemented using the ReAct (Reason + Act) paradigm, allowing the LLM to iteratively reason, take actions through tools, and refine its conclusions based on observations.

The goal was to develop all components from scratch, including the reasoning loop, tool system, SQL validation, and prompt scaffolding, to understand the internal structure of modern AI agents rather than using pre-built frameworks.

2. SYSTEM DESIGN

Architecture Overview

User Query -> LLM Thought -> Action -> Observation -> LLM Thought -> Final Answer

The system consists of the following modules:

- SQLAgent: main reasoning and control loop.
- ToolRegistry: collection of executable database tools.
- safe_sql(): ensures SQL safety by blocking non-SELECT operations.
- SYSTEM_PROMPT: defines agent rules, structure, and tool descriptions.
- Retry mechanism: handles transient API failures.

Tools Implemented

- list tables(): Lists all tables in the SQLite database.
- describe table(table_name): Displays column names, types, and row count.
- query database(query): Executes a validated SELECT query and returns results.

SQL Validation

A custom validation function ensures that only safe read-only SQL statements are executed.

3. IMPLEMENTATION

The agent was implemented in a Google Colab Notebook using:

- google-generativeai - Gemini API client for LLM reasoning.
- sqlite3 - Python's built-in database engine.
- tabulate - for neatly formatted table outputs.

Sample Database

Table: students(id INTEGER, name TEXT, grade REAL)

Sample Data: (1, Alice, 85.5), (2, Bob, 78.0), (3, Clara, 92.3)

4. RESULTS

Query: "List all students and their grades."

Output:

Alice: 85.5, Bob: 78.0, Clara: 92.3

Test Cases:

1. List all tables -> ['students']

2. Describe table -> Displays schema.

3. Filter grade > 80 -> Alice, Clara.

4. Invalid DELETE -> Refused safely.

5. DISCUSSION

The agent effectively demonstrates controlled interaction between reasoning and action. It interprets vague queries, plans reasoning steps, executes SQL commands safely, and responds accurately.

6. REFLECTION

This assignment deepened understanding of:

- ReAct architecture and tool-based reasoning.
- Framework-free agent design.
- Prompt grounding and SQL safety.

7. CONCLUSION

A fully functional ReAct SQL Database Agent was successfully implemented. It integrates Gemini reasoning, structured database access, safe SQL validation, and explainable reasoning traces, all without relying on external frameworks.

8. REFERENCES

- Yao, Shunyu et al., "ReAct: Synergizing Reasoning and Acting in Language Models." 2022.
- Google AI Studio - Gemini API Documentation.
- Python Standard Library Documentation.

Appendix A - One-Line Command

agent.run("List all students and their grades.")