Assignment: The "No-SQL" Database Agent

Overview

In this assignment, you will bridge the gap between unstructured natural language and structured data storage. You will build an AI Agent capable of managing a **SQLite database** purely through conversation.

The user (you) will speak normal English (e.g., *"Hire Alice as a Manager"*), and the LLM must determine which Python functions to call to actually modify the database file.

The Scenario

Imagine you are building a backend system for a company HR department, but the manager doesn't know how to code or write SQL. They just want to chat with a bot to manage employees. Your agent will act as the translator between their requests and the raw database.

Technical Requirements

1. The Database (The "World")

You must use **SQLite** (built into Python) to create a persistent local file (e.g., `company.db`).

- The data structure is up to you, but it must be capable of storing at least: **Names, Roles/Departments, and Salaries.**

2. The Tools (The "Hands")

You need to define and implement Python functions that perform specific SQL operations. These functions will be exposed to the LLM. You must implement at least the following capabilities:

- **Initialization:** A tool to create the necessary table(s) if they don't exist.
- **Insertion:** A tool to add a new employee record.
- **Deletion:** A tool to remove an employee (e.g., by name or ID).
- **Retrieval:** A tool to query data (e.g., "Find all Managers" or "Get Alice's salary").

Constraint: Your Python functions should execute the actual SQL commands. The LLM should *not* simply be asked to write SQL code itself; it should call your specific functions (e.g., `add_employee(name="Alice", salary=50000)`) to ensure safety and structure.

3. The Logic Loop (The "Brain")

Your main script must implement the standard Function Calling loop:

1. **Input:** Take a user string from the terminal.

2. **Decision:** Send the input + tool definitions to the LLM.
3. **Execution:** If the LLM requests a tool call, your script must execute the corresponding Python function against the SQLite database.
4. **Feedback:** Pass the return value of the function (e.g., "Success: Row ID 5 created" or "Error: User not found") back to the LLM.
5. **Response:** The LLM generates a final natural language confirmation for the user.

Example Interaction Flow

- **User:** "We just hired John Doe. He's an Engineer making $80k."
- **Agent (Internal Thought):** *I need to call `insert_employee` with arguments: name='John Doe', role='Engineer', salary=80000.*
- **System:** Executes SQL `INSERT`.
- **Agent (Output):** "I've added John Doe to the database as an Engineer."
- **User:** "Actually, fire him."
- **Agent (Internal Thought):** *I need to call `delete_employee`.*
- **System:** Executes SQL `DELETE`.
- **Agent (Output):** "John Doe has been removed from the records."

Deliverables

`Python file/ Jupyter Notebook / Colab Link -` Your complete source code.

Grading Criteria

- **Tool Definition:** Are the tool schemas (JSON) defined correctly so the LLM understands its parameters?
- **Persistence:** Does the script actually modify the `.db` file? (If I restart the script, is the data still there?)
- **Error Handling:** What happens if the user asks to delete someone who doesn't exist? The agent should handle this gracefully, not crash.

Due Date: 8 days from assignment date