



Search

Write



C

Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)

Build your First SQL Database Agent with LangChain

Lawrence Knight · [Follow](#)

4 min read · Aug 21, 2023

156

3

A step-by-step guide to building a LangChain enabled SQL database question answering agent.

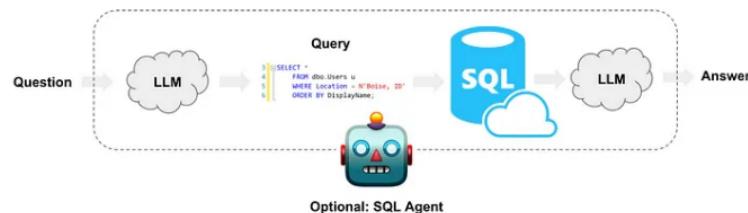


Table of Contents

1. Introduction

2. Useful Resources

3. Agent Code

- Configuration
- Import Packages
- Instantiate the Database
- Instantiate the LLM
- Create the Agent
- Test the Agent

4. Conclusion

Introduction

In this tutorial, we will walk through step-by-step, the creation of a LangChain enabled, large language model (LLM) driven, agent that can use a SQL database to answer questions. In practice, this functionality has the potential to be incredibly usefully. If we take customer servicing as an example, most organisations hold their customer data in SQL databases. Developing automated customer servicing agents that allow customers to query and update their data presents an opportunity to significantly reduce customer servicing costs.

There are a number of different way to create SQL question answering services using LangChain. In this example we will be using LangChain's SQL Agent. Two fo the benefits of the SQL Agent over other approaches are:

1. The ability to answer questions based on the databases' schema as well as on the databases' content (like describing a specific table).

2. The ability to recover from errors by running a generated query, catching the traceback and regenerating it correctly.

Useful Resources

The code in this tutorial draws heavily from the LangChain documentation, links to which are provided below.

- https://python.langchain.com/docs/use_cases/sql
- https://python.langchain.com/docs/integrations/toolkits/sql_database

Agent Code

The code in this notebook is complete and if copied into a notebook will produce a working agent.

Configuration

To run this code you will need an OpenAI account with a paid for subscription to access their large language model APIs.

```
# insert your API key here
openai_api_key = "<>"
```

Import Packages

```
from langchain.utilities import SQLDatabase
from langchain.llms import OpenAI
from langchain.agents import create_sql_agent
from langchain.agents.agent_toolkits import SQLDatabaseToolkit
from langchain.agents.agent_types import AgentType
```

Instantiate the Database

Before we can instantiate the database we need to go to the SQLite website and download the Chinook sample database. You can download it [here](#). Once download, save it to the same directory as your notebook.

```
db = SQLDatabase.from_uri("sqlite:///chinook.db")
```

```
# check that the database has been instantiated correctly
db.get_usable_table_names()
```

You should see a list of table names.

Instantiate the LLM

```
llm = OpenAI(temperature=0, verbose=True, openai_api_key=openai_api_key)
```

Create the Agent

The agent uses the ReACT prompting technique, introduced in the paper “[REACT: Synergizing Reasoning and Acting in Language Models](#)”, by a team of researchers from Google Research and Princeton University. ReACT combines Chain-of-Thought reasoning and action plan generation integrating the use of tools into the reasoning process. If you don’t have time to read the whole paper you can find a summary [here](#).

```
agent_executor = create_sql_agent(
    llm=llm,
    toolkit=SQLDatabaseToolkit(db=db, llm=llm),
    verbose=True,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
)
```

Test the Agent

Example 1 – Running Queries

In this example we demonstrate how the agent can query a database to answer questions.

```
agent_executor.run(
    "How many employees are there?"
)
```

```
> Entering new AgentExecutor chain...
Action: sql_db_list_tables
Action Input:
Observation: albums, artists, customers, employees, genres, invoice_items, invoices, media_types, playlist_tracks
Thought: The table I need is employees.
Action: sql_db_schema
Action Input: employees
Observation:
CREATE TABLE employees (
    "EmployeeId" INTEGER NOT NULL,
    "LastName" NVARCHAR(20) NOT NULL,
    "FirstName" NVARCHAR(20) NOT NULL,
    "Title" NVARCHAR(30),
    "ReportsTo" INTEGER,
    "BirthDate" DATETIME,
    "HireDate" DATETIME,
    "Address" NVARCHAR(70),
    "City" NVARCHAR(40),
    "State" NVARCHAR(40),
    "Country" NVARCHAR(40),
    "PostalCode" NVARCHAR(10),
    "Phone" NVARCHAR(24),
    "Fax" NVARCHAR(24),
    "Email" NVARCHAR(60),
    PRIMARY KEY ("EmployeeId"),
    FOREIGN KEY("ReportsTo") REFERENCES employees ("EmployeeId")
)
/*
3 rows from employees table:
EmployeeId   LastName   FirstName   Title   ReportsTo   BirthDate   HireDate   Address   City
1           Adams      Andrew      General Manager   None   1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW
Edmonton
2           Edwards    Nancy      Sales Manager   1       1958-12-08 00:00:00  2002-05-01 00:00:00  925 8 Ave SW
Calgary AB
3           Peacock   Jane      Support Agent  2       1973-08-29 00:00:00  2002-04-01 00:00:00  1111 6 Ave
SW
Canada T2P 2T3 +1 (403) 262-3443 +1 (403) 262-3322 nancy@chinookcorp.com
*/
Thought: I should query the employees table for the number of employees.
Action: sql_db_query
Action Input: SELECT COUNT(*) FROM employees
Observation: [8]
Thought: I now know the final answer.
Final Answer: There are 8 employees.

> Finished chain.
'There are 8 employees.'
```

We can observe the cycle of thought, action, observation that the agent goes through to get the answer. The prompt that illicits this human like reasoning is simple and is given below. I find it incredible that through this simple direction complex reasoning behaviour emerges.

```
FORMAT_INSTRUCTIONS = """
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of {{tool_names}}
Action Input: the input to the action
Observation: the output of the action
"""

# Example 1 - Q: How many employees are there?
# A: SELECT COUNT(*) FROM employees
# O: 8
```

```

observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question"""

```

Example 2— Describing a Table

In this example we demonstrate how the agent can describe a table in the database.

```
agent_executor.run("Describe the playlisttrack table")
```

```

> Entering new AgentExecutor chain...
Action: sql_db_list_tables
Action Input:
Observation: albums, artists, customers, employees, genres, invoice_items, invoices, media_types, playlist_track, p
laylists, tracks
Thought: The playlisttrack table looks relevant, so I should query the schema of that table.
Action: sql_db_schema
Action Input: playlist_track
Observation:
CREATE TABLE playlist_track (
    "PlaylistId" INTEGER NOT NULL,
    "TrackId" INTEGER NOT NULL,
    PRIMARY KEY ("PlaylistId", "TrackId"),
    FOREIGN KEY("PlaylistId") REFERENCES playlists ("PlaylistId"),
    FOREIGN KEY("TrackId") REFERENCES tracks ("TrackId")
)

/*
3 rows from playlist_track table:
PlaylistId   TrackId
1            3402
1            3389
1            3390
*/
Thought: I now know the final answer
Final Answer: The playlist_track table contains two columns, PlaylistId and TrackId, which are both integers and form a primary key. It also has two foreign keys, one to the playlists table and one to the tracks table.

> Finished chain.
The playlist_track table contains two columns, PlaylistId and TrackId, which are both integers and form a primary key. It also has two foreign keys, one to the playlists table and one to the tracks table.

```

Example 3— Recovering from an Error

In this example, the agent is able to recover from an error after initially trying to access an attribute that doesn't exist.

```
agent_executor.run("Who are the top 3 best selling artists?")
```

```

> Entering new AgentExecutor chain...
Action: sql_db_list_tables
Action Input:
Observation: albums, artists, customers, employees, genres, invoice_items, invoices, media_types, playlist_track, p
laylists, tracks
Thought: I should query the schema of the albums and artists tables.
Action: sql_db_schema
Action Input: albums, artists
Observation:
CREATE TABLE albums (
    "AlbumId" INTEGER NOT NULL,
    "Title" NVARCHAR(160) NOT NULL,
    "ArtistId" INTEGER NOT NULL,
    PRIMARY KEY ("AlbumId"),
    FOREIGN KEY("ArtistId") REFERENCES artists ("ArtistId")
)

/*
3 rows from albums table:
AlbumId Title   ArtistId
1      For Those About To Rock We Salute You  1
2      Balls to the Wall   2
3      Restless and Wild   2
*/
CREATE TABLE artists (
    "ArtistId" INTEGER NOT NULL,
    "Name" NVARCHAR(120),
    PRIMARY KEY ("ArtistId")
)

/*
3 rows from artists table:
ArtistId Name
1       AC/DC
2       Accept
3       Aerosmith
*/
Thought: I should query the albums table to get the top 3 best selling artists.
Action: sql_db_query
Action Input: SELECT a.Name, COUNT(*) AS 'Number of Albums Sold' FROM albums a JOIN artists b ON a.ArtistId = b.Art
istId GROUP BY a.Name ORDER BY COUNT(*) DESC LIMIT 3;
Observation: Error: (sqlite3.OperationalError) no such column: a.Name
[SQL: SELECT a.Name, COUNT(*) AS 'Number of Albums Sold' FROM albums a JOIN artists b ON a.ArtistId = b.ArtistId GR
UP BY a.Name ORDER BY COUNT(*) DESC LIMIT 3]
(Background on this error at: https://sqlalche.me/e/20/g3q8)
Thought: I should double check my query before executing it.
Action: sql_db_query_checker
Action Input: SELECT a.Name, COUNT(*) AS 'Number of Albums Sold' FROM albums a JOIN artists b ON a.ArtistId = b.Art
istId GROUP BY a.Name ORDER BY COUNT(*) DESC LIMIT 3;
Observation:
SELECT a.Name, COUNT(*) AS 'Number of Albums Sold'
FROM albums a

```

```

JOIN artists b ON a.ArtistId = b.ArtistId
GROUP BY a.Name
ORDER BY COUNT(*) DESC
LIMIT 3;
Thought: I should query the albums table to get the top 3 best selling artists.
Action: sql_db_query
Action Input: SELECT b.Name, COUNT(*) AS 'Number of Albums Sold' FROM albums a JOIN artists b ON a.ArtistId = b.ArtistId GROUP BY b.Name ORDER BY COUNT(*) DESC LIMIT 3;
Observation: [('Iron Maiden', 21), ('Led Zeppelin', 14), ('Deep Purple', 11)]
Thought: I now know the final answer.
Final Answer: The top 3 best selling artists are Iron Maiden, Led Zeppelin, and Deep Purple.

> Finished chain.
'The top 3 best selling artists are Iron Maiden, Led Zeppelin, and Deep Purple.'

```

Conclusion

This technology is exceptionally powerful! Personally I love to inspect the chains of reasoning and action and marvel at where this type of technology might take us. In the immediate present this type of functionality has much potential in the automation of business processes. Based on my experimentation with this tech the reliability of this approach begins to break down when:

1. the database structure is complex;
2. table names are ambiguous;
3. the questions asked can not be answered through a single simple query.

This can be managed by investing time in adding additional information to the prompt, such as:

1. provide a database description for your specific use case.
2. provide descriptions of fields in the db.
3. hardcoding a few examples of questions and their corresponding SQL queries in the prompt.

Have fun experimenting with your own usecases:)

References

- https://python.langchain.com/docs/use_cases/sql
- https://python.langchain.com/docs/integrations/toolkits/sql_database

Want to get in touch?

Follow on Medium and [Linkedin](#) for summaries of papers at the cutting edge of machine learning and intelligence plus more general thoughts on how research is progressing in the field.

Large Language Models

NLP

Langchain

Artificial Intelligence

Data Science

🕒 156 🗃 3

⬇️ ⬆️ ⋮



Written by Lawrence Knight

Follow



79 Followers

An artificial intelligence enthusiast with a broad range of interests. Projects in reinforcement learning are my current focus.

More from Lawrence Knight



Lawrence Knight in Dev Genius

Document Tagging with LangChain

A step-by-step guide to building a LangChain enabled document tagging service.

7 min read · Aug 23, 2023

8

+

...



Lawrence Knight in Dev Genius

Build your First Conversational Document Retrieval Agent using...

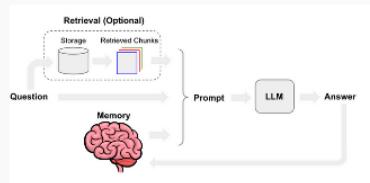
A step-by-step guide to building a Llama 2 powered, LangChain enabled conversational...

9 min read · Aug 25, 2023

101

+

...



Lawrence Knight in Dev Genius

Build your First Conversational Document Retrieval Agent using...

A step-by-step guide to building a LangChain enabled conversational document retrieval...

5 min read · Aug 18, 2023

48

+

...



Lawrence Knight

Automatic Chain-of-Thought Prompting in Large Language...

Introduction

4 min read · Aug 1, 2023

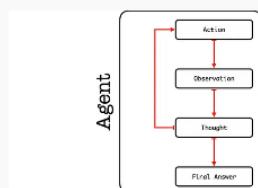
5

+

...

[See all from Lawrence Knight](#)

Recommended from Medium



Saumitra Joshi

Langchain (Agents, Tools, Chains & Memory) for utilizing the full...

Introduction

19 min read · Nov 8, 2023



Rohit Raj

Part 6: How to chat with SQL database with Langchain and...

Use Langchain along with SQLITE3

3 min read · Dec 18, 2023

177



+

...

56



+

...

Lists



Natural Language Processing

1349 stories · 828 saves



Predictive Modeling w/ Python

20 stories · 1060 saves



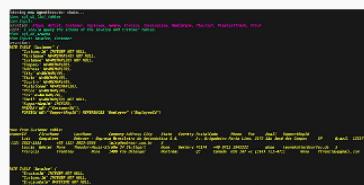
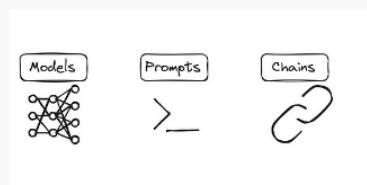
AI Regulation

6 stories · 401 saves



ChatGPT prompts

47 stories · 1366 saves



Shivansh Kaushik

Talk to your Database using RAG and LLMs

What is RAG?

5 min read · Oct 12, 2023

87



+

...

2



+

...

Gabriel Gomes, PhD

Deep dive in LangChain for Relational DB interactions

In the past months, several contents have been produced related to using LangChain...

10 min read · Nov 11, 2023



Arun Shankar in Google Cloud - Community

Architectural Patterns for Text-to-SQL: Leveraging LLMs for...

TLDR: This article delves into the Text-to-SQL domain, demonstrating the growing reliance...

36 min read · Nov 12, 2023

459



+

...

1.4K



+

...

Yvann in Better Programming

Build a Chatbot on Your CSV Data With LangChain and OpenAI

Chat with your CSV file with a memory chatbot —Made with Langchain...

5 min read · Jun 3, 2023

See more recommendations