Query SQL database using natural language

Overview

In today's data-driven world, professionals across various industries often need to extract insights from large SQL databases. Traditionally, this process involved writing complex SQL queries, performing extensive data manipulation, and creating visual dashboards to interpret results. These tasks can be time-consuming and require a high level of technical expertise.

However, with advancements in natural language processing and AI technologies, querying SQL databases has become much more intuitive and accessible. Imagine being able to ask questions like, "What are the sales trends for the past year?" or "Which products have the highest return rates?" and instantly receive detailed, visualized responses. This guided project guides you through setting up an agent that lets you query a MySQL database using natural language, simplifying the data analysis process.

Objective

In the project, you:

- **Integrate natural language processing**: Use tools like Langchain and large language models (LLM) to interpret natural language queries.
- **Execute SQL queries from natural language**: Translate natural language questions into SQL queries to fetch relevant data from the MySQL database.

Set up a virtual environment

Begin by creating a virtual environment. Using a virtual environment lets you manage dependencies for different projects separately, avoiding conflicts between package versions.

In the terminal of your Cloud IDE, ensure that you are in the path `/home/project`, then run the following commands to create a Python virtual environment.

```
pip install virtualenv
virtualenv my_env # create a virtual environment named my_env
source my_env/bin/activate # activate my_env
```

Copied! Executed!

Install necessary libraries

To ensure a seamless execution of the scripts, and considering that certain functions within these scripts rely on external libraries, it's essential to install some prerequisite libraries before you begin.

- `ibm-watsonx-ai` and `ibm-watson-machine-learning`: The IBM Watson Machine Learning package integrates powerful IBM LLM models into the project.
- `langchain`, `langchain-ibm`, and `langchain-experimental`: This library is used for relevant features from Langchain.
- `mysql-connector-python`: This library is used as a MySQL database connector.

Run the following commonds in your terminal (with the `my_env` prefix) to install the packages.

```
python3.11 -m pip install ibm-watsonx-ai==1.0.4 \
ibm-watson-machine-learning==1.0.357 \
langchain==0.2.1 \
langchain-ibm==0.1.7 \
langchain-experimental==0.0.59 \
mysql-connector-python==8.4.0
```

Copied! Executed!

Instantiate a MySQL database

Because this lab focuses on querying a MySQL database using natural language, you must instantiate a MySQL server, and then create a sample database in the server.

Create MySQL server

To create a MySQL server in Cloud IDE, click the following button.

Open and Start MySQL in IDE

After you click the button, you see that there is a MySQL service on the right. Click **Start** to start the MySQL server.

It might take approximately 10-15 seconds. When it's showing active, it means that the server is ready to be used. If you see any error messages, please refresh the page and try again.

You have created a MySQL server. Let's test it to see if it can run successfully.

Click **MySQL CLI** so that you can interact with the server in the terminal. It opens a new tab in the terminal showing something similar to the following image with the `mysql` prefix at the front.

THis means that you have successfully connected with the server. Now, let's input an SQL query to test it. For example, the following command shows all databases in the server.

```
SHOW DATABASES;
```

Copied!

To run it, press `enter/return` on your keyboard. If it runs successcully, it returns an output like shown in the following image.

Congratulations, the server is working correctly. Now, let's create a sample database to use.

Create Chinook database

In this lab, you use the Chinook database as an example.

Introduction to the Chinook database

The Chinook data model represents a digital media store, including tables for artists, albums, media tracks, invoices, and customers.

- Media-related data was created by using real data from an Apple iTunes library.
- Customer and employee information was created by using fictitious names and addresses that can be located on Google maps, and other well-formatted data (phone, fax, email, and so on).
- Sales information was auto generated using random data for a four-year period.

The Chinook sample database includes:

- 11 tables
- A variety of indexes, primary and foreign key constraints
- Over 15,000 rows of data

For details, the following image shows the entity relationship diagram of the Chinook data model.

Retrieve the database creation code

The database creation code has been prepared for you. Run the following code in the terminal to retrieve the SQL file from the remote.
Note: Run the code in the terminal with the `(my_env)` prefix instead of the `mysql` prefix.

```
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/Mauh_UvY4eK2SkcHj_b8Tw/chinook-mysql.sql
```

Copied! Executed!

After it runs successfully, you see an SQL file called `PROJECT`.

Run the sql file

Now, you must excute the SQL file to create the database.

At the terminal with the `mysql` prefix, enter the following command.

```
SOURCE chinook-mysql.sql;
```

Copied!

After it's finished, test to see if the database was created successfully by entering the following command.

```
SHOW DATABASES;
```

Copied!

If the database was successfully created, you see the `Chinook` database in your list of databases.

Let's run some sample SQL commands to interact with the `Chinook` database. For example, suppose that you want to know how many albums the `Chinook` database contains. To find this information, you could run the following SQL command.

```
USE Chinook;
SELECT COUNT(*) FROM Album;
```

Copied!

The previous command, when copied and pasted into the terminal with the `mysql` prefix should give you an answer of 347.

Instantiate an LLM

This section guides you through the process of instantiating an LLM by using the watsonx.ai API. This section uses the `mistralai/mixtral-8x7b-instruct-v01` model. To find other foundational models that are available on watsonx.ai, refer to Foundation model library and fm_model.

Create the LLM

Click the following button to create an empty Python file that is named `sql_agent.py`.

Open **sql_agent.py** in IDE

The following code creates an LLM object by using the watsonx.ai API. Copy and paste it to the `sql_agent.py` file, then go to **File –> Save**.

```python
# Use this section to suppress warnings generated by your code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai.foundation_models.utils.enums import ModelTypes
from ibm_watson_machine_learning.foundation_models.extensions.langchain import WatsonxLLM
from langchain_community.utilities.sql_database import SQLDatabase
from langchain_community.agent_toolkits import create_sql_agent

model_id = 'mistralai/mixtral-8x7b-instruct-v01'

parameters = {
    GenParams.MAX_NEW_TOKENS: 256,  # This controls the maximum number of tokens in the generated output
    GenParams.TEMPERATURE: 0.5, # This randomness or creativity of the model's responses
}

credentials = {
    "url": "https://us-south.ml.cloud.ibm.com"
}

project_id = "skills-network"

model = ModelInference(
    model_id=model_id,
    params=parameters,
    credentials=credentials,
    project_id=project_id
)

mixtral_llm = WatsonxLLM(model = model)

print(mixtral_llm.invoke("What is the capital of Ontario?"))
```

Copied!

Note that in addition to creating the LLM object the previous code includes the sample query ‘What is the capital of Ontario?’. This query was included to test whether the model is being correctly loaded by the script.

Test the model

Run the following command in the terminal with the `my_env` prefix.

```
python3 sql_agent.py
```

Copied! Executed!

If you are successful, you see a response in the terminal that’s similar to the following response. Note that you will likely not get an identical response because the LLM was instantiated with a temperature of 0.5 ensuring that there is quite a bit of randomness in the model’s responses.

Great, the LLM is ready to be used!

Build the database connector

Building the database connector is simple. It requires just two lines of code:

```
mysql_uri = 'mysql+mysqlconnector://{mysql_username}:{mysql_password}@{mysql_host}:{mysql_port}/{database_name}'
db = SQLDatabase.from_uri(mysql_uri)
```

Copied!

However, from the code, you see that there are some MySQL server-related parameters that are missing. Specifically, you must define a `mysql_username`, a `mysql_password`, a `mysql_host`, a `mysql_port`, and a `database_name`. Where can you find the values of these parameters?

Go to the `MySQL` service. Then, click the `Connection Information` tab. Under this tab, you find almost all of the necessary parameter values, except for the `database_name`, which is the database you created before, `Chinook`.

Replace the connection information in the following code with the values that you get from your MySQL server.

```
mysql_username = 'root'  # Replace with your server connect information
mysql_password = 'zQTLH3HB0q3ahCuAaDcAwdlb' # Replace with your server connect information
mysql_host = '172.21.52.20' # Replace with your server connect information
mysql_port = '3306' # Replace with your server connect information
database_name = 'Chinook'
mysql_uri = f'mysql+mysqlconnector://{mysql_username}:{mysql_password}@{mysql_host}:{mysql_port}/{database_name}'
db = SQLDatabase.from_uri(mysql_uri)
```

Copied!

You should add above code, with the connection information replaced by the values of your MySQL server, by appending them to the `sql_agent.py` file.

Also, delete the following line from the `sql_agent.py` file. This was the last line in the file prior to appending the two lines previously.

```
print(mixtral_llm.invoke("What is the capital of Ontario?"))
```

Copied!

After adding the two lines for the database connector and deleting the `print()` line, save `sql_agent.py`.

> The following code is the complete code until now. Note that you should amend the `mysql_uri` = line as necessary.

▶ Code

Create the SQL query agent

You'll use the `create_sql_agent` from LangChain to interact with your SQL database. The primary benefits of using this SQL agent include:

- **Answering questions** based on the database's schema and content, such as describing specific tables.
- **Error recovery** by executing a generated query, capturing any tracebacks, and regenerating the query if necessary.
- **Multiple queries** to the database to thoroughly answer user questions.
- **Token efficiency** by retrieving schemas only from pertinent tables.

Creating an SQL query agent requires the addition of just one line of code. Append the following line to the `sql_agent.py` file.

```
agent_executor = create_sql_agent(llm=mixtral_llm, db=db, verbose=True)
```

Copied!

The final thing that you must do is to actually query your SQL agent. The following code asks the SQL agent to find the number of albums in the database by using plain English. Append these lines to the `sql_agent.py` file and save the file.

```
agent_executor.invoke(
    "How many Album are there in the database?"
)
```

Copied!

The following code is the completed `sql_agent.py` file. Note that you would have to modify the `mysql_uri =` line to point to your MySQL database.

▶ Code

To use your agent, go to the terminal with the `my_env` prefix, and run the following command.

```
python3 sql_agent.py
```

Copied! Executed!

If the agent runs successfully, you see a response that is similar to the following response in your terminal.

Because you set `verbose=True` in the code, you not only see the `Final Answer`, but also all of the actions that are undertaken by the LLM and the SQL code the LLM generated to reach its `Final Answer`. If everything worked correctly, the agent should tell you that there are 347 albums in the database, the exact same result found by using SQL code.

You can use the agent to run different queries, not just the query you saw previously. For instance, if you replace the query in the `sql_agent.py` file with “Describe the database for me.”, you can expect the Agent to produce output similar to the following image.

Isn’t it amazing?

Congratulations, you have finished the project!

Author(s)

Kang Wang
Kang Wang is a Data Scientist in the IBM. He is also a PhD Candidate in the University of Waterloo.

Wojciech Fulmyk
Wojciech “Victor” Fulmyk is a Data Scientist at IBM. He is also a PhD Candidate in Economics in the University of Calgary.

Ricky Shi
Ricky Shi is a Data Scientist at IBM.

Changelog

Date	Version	Changed by	Change Description
2024-06-09	0.1	Kang Wang	Initial version created
2024-06-14	0.2	Wojciech Fulmyk	Lab edited: Grammar, clarity, and minor code fixes