Build a Natural Language SQL Agent

Overview

In the modern business landscape, organizations face an ongoing challenge of extracting meaningful insights from vast SQL databases. The conventional approach involves crafting complex SQL queries, manipulating data, and building visualization tools—tasks that demand considerable time and specialized technical knowledge.

The emergence of **AI** and **natural language processing** technologies has transformed how we interact with databases. Instead of mastering complex SQL syntax, users can now simply **ask questions in everyday language** such as ***Show me last year's revenue patterns*** or ***Which items are most frequently returned?*** and get instant, visualized answers. This project will guide you through setting up an AI-powered system that converts natural language into MySQL database queries, making data analysis accessible to everyone on your team.

Objective

In this project, you will:

- **Integrate natural language processing**: Use AI tools 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.

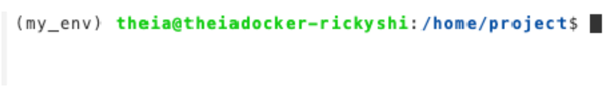
Set up a virtual environment

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

In your Cloud IDE **terminal**, 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
```

You will see the `(my_env)` prefix in your terminal. This prefix indicates that the virtual environment is **active**.



```
(my_env) theia@theiadocker-rickyshi:/home/project$
```

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 that you 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-community`: This library is used for relevant features from LangChain.
- `mysql-connector-python`: This library is used as a MySQL database connector.

Run the following commands 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
```

Instantiate a MySQL database

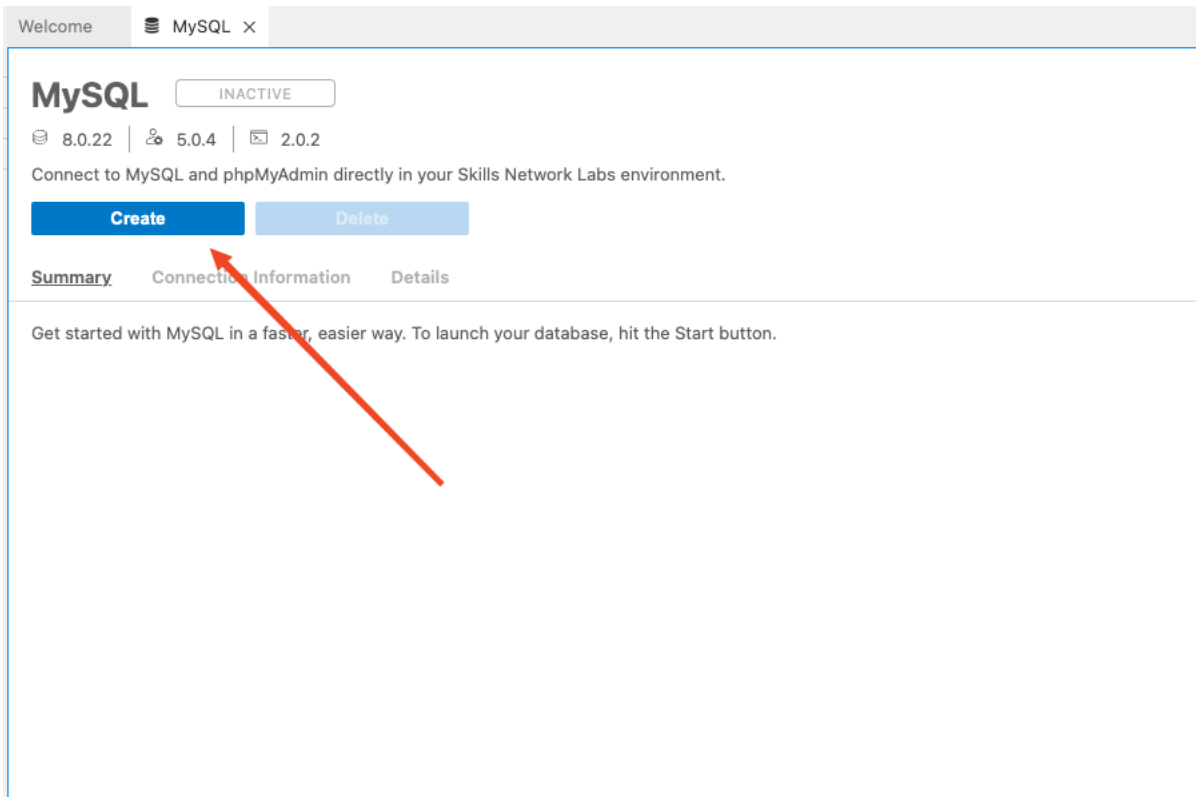
Because this lab focuses on querying a **MySQL** database using natural language, you must instantiate a MySQL server, and then create a sample database in the server.

Create a MySQL server

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

Open and Start MySQL in IDE

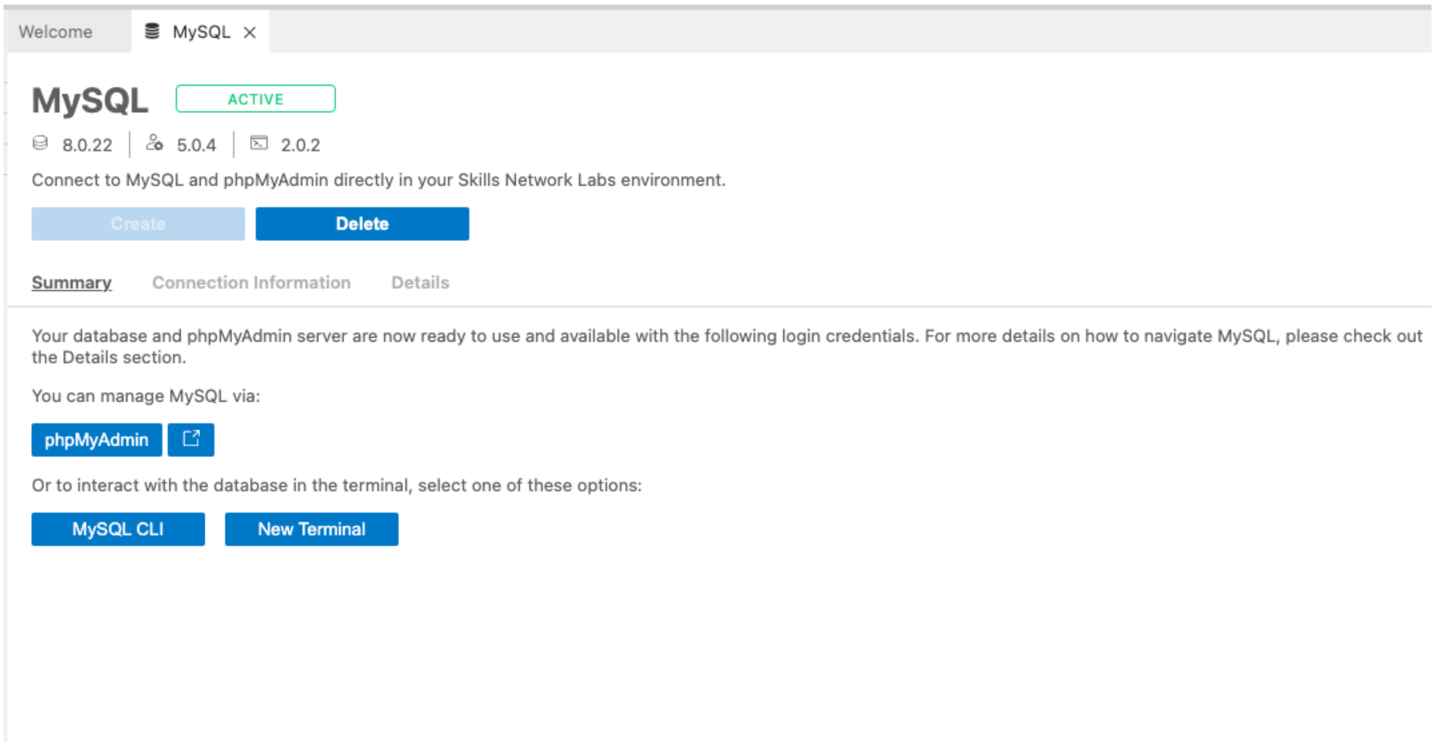
After you click the button, you'll see a MySQL service on the right. Click **Create** to start the MySQL server.



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

Get started with MySQL in a faster, easier way. To launch your database, hit the Start button.

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



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

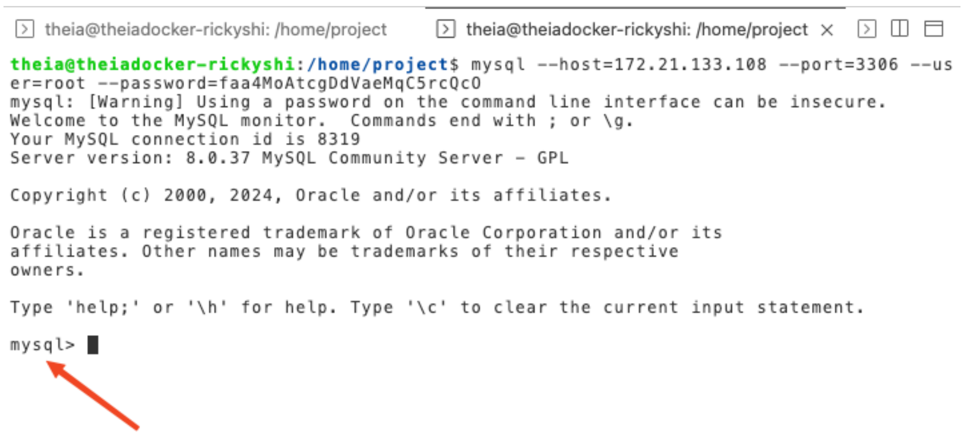
Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section.

You can manage MySQL via:

Or to interact with the database in the terminal, select one of these options:

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

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

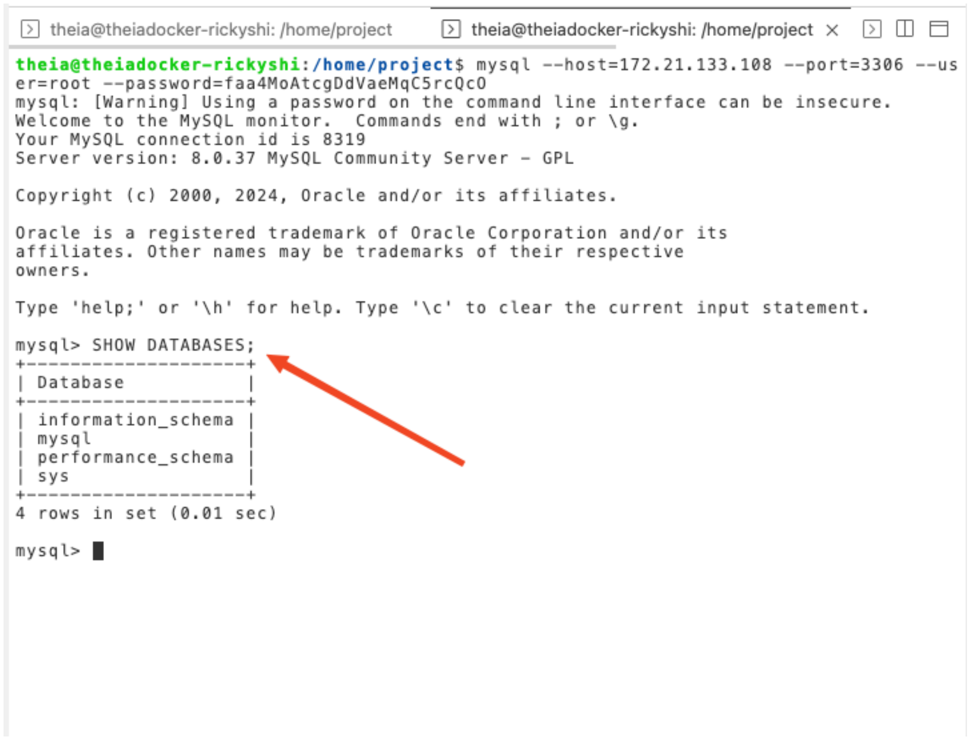

theia@theiadocker-rickyshi: /home/project
theia@theiadocker-rickyshi: /home/project

```
theia@theiadocker-rickyshi:/home/project$ mysql --host=172.21.133.108 --port=3306 --us
er=root --password=faa4MoAtcgDdVaeMqC5rcQcO
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8319
Server version: 8.0.37 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

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;
```

Please copy and paste the above command into the terminal with the `mysql` prefix, and press `enter/return` on your keyboard. If it runs successfully, it returns an output similar to the following image.


theia@theiadocker-rickyshi: /home/project
theia@theiadocker-rickyshi: /home/project

```
theia@theiadocker-rickyshi:/home/project$ mysql --host=172.21.133.108 --port=3306 --us
er=root --password=faa4MoAtcgDdVaeMqC5rcQcO
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8319
Server version: 8.0.37 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)

mysql> 
```

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

Create the Chinook database

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

Introduction to the Chinook database

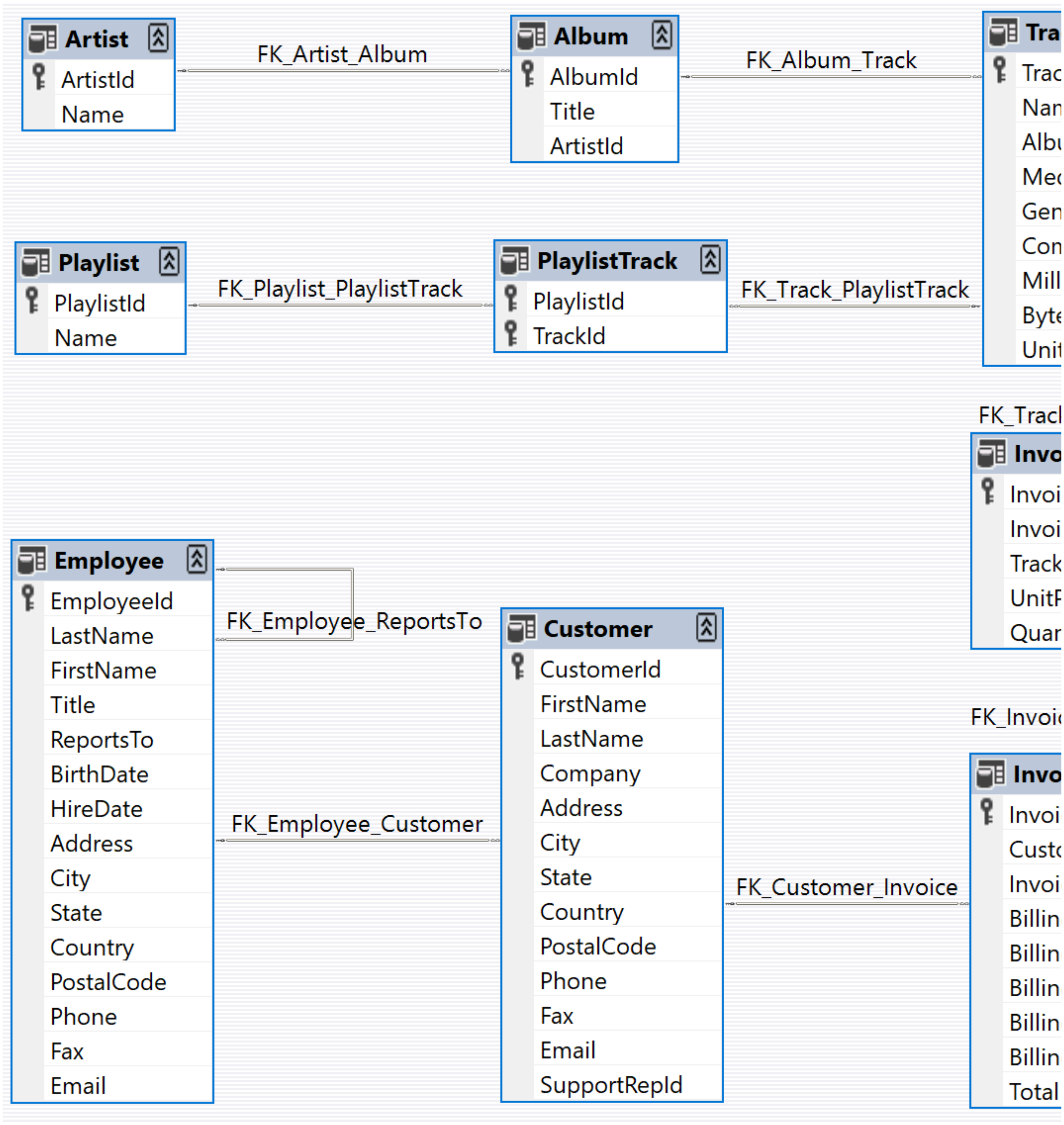
The Chinook database models a comprehensive digital media store ecosystem, featuring interconnected tables that track artists, albums, media tracks, invoices, and customer information. Here's what makes it unique:

- The media catalog contains authentic data sourced directly from an Apple iTunes library
- Customer and employee records use carefully crafted fictional data, including Google Maps-verified addresses and properly formatted contact details (phone numbers, fax, email addresses)
- The sales data spans a 4-year period and was generated programmatically with randomized but realistic values

Key characteristics of the database:

- Comprehensive structure with **11** distinct tables
- Robust data integrity through indexes and primary/foreign key relationships
- Rich dataset containing more than **15,000** records

Below you'll find the **entity relationship diagram (ERD)** that illustrates how the different components of the Chinook database connect and interact with each other.



Retrieve the database creation code

The database creation code has been prepared for you.

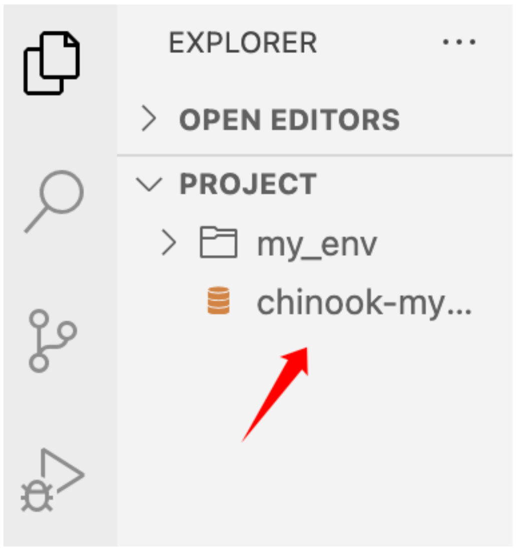
Before you run the code, please ensure that you are in the path `/home/project` in the terminal with the `(my_env)` prefix.

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
```

After it runs successfully, you'll see an **SQL file** in `PROJECT`.



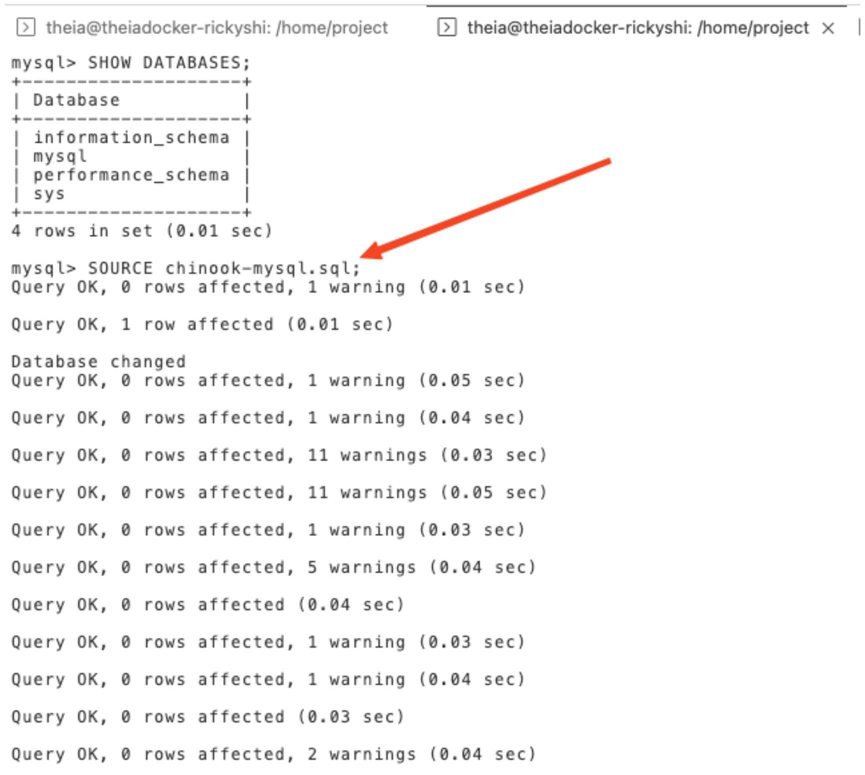
Run the SQL file

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

Note: Run the code in the terminal with the `mysql` prefix instead of the `(my_env)` prefix.

At the terminal with the `mysql` prefix, copy and paste the following command.

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


theia@theiadocker-rickyshi: /home/project
theia@theiadocker-rickyshi: /home/project

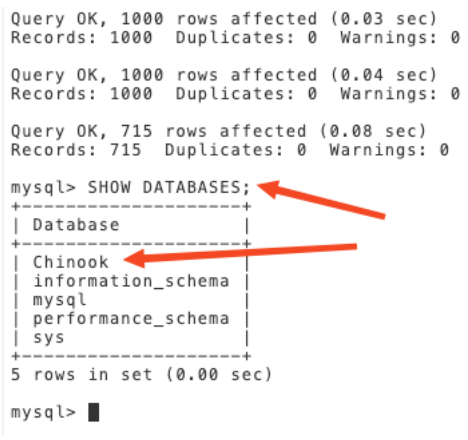
```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)

mysql> SOURCE chinook-mysql.sql;
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 1 row affected (0.01 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.05 sec)

Query OK, 0 rows affected, 1 warning (0.04 sec)

Query OK, 0 rows affected, 11 warnings (0.03 sec)

Query OK, 0 rows affected, 11 warnings (0.05 sec)

Query OK, 0 rows affected, 1 warning (0.03 sec)

Query OK, 0 rows affected, 5 warnings (0.04 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected, 1 warning (0.03 sec)

Query OK, 0 rows affected, 1 warning (0.04 sec)

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected, 2 warnings (0.04 sec)
```

After it runs successfully, you'll see the `Chinook` database in your list of databases.

To check this, you can run the following command.

```
SHOW DATABASES;
```

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



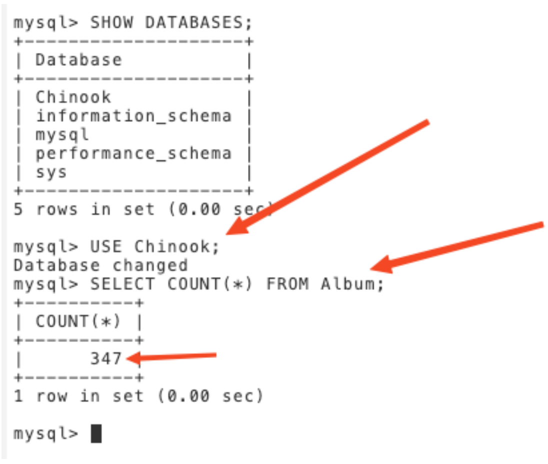
```
Query OK, 1000 rows affected (0.03 sec)
Records: 1000  Duplicates: 0  Warnings: 0

Query OK, 1000 rows affected (0.04 sec)
Records: 1000  Duplicates: 0  Warnings: 0

Query OK, 715 rows affected (0.08 sec)
Records: 715  Duplicates: 0  Warnings: 0

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| Chinook            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> 
```

Congratulations, the database was created successfully.

Now let's run some sample SQL commands to interact with the `Chinook` database. For example, suppose 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;
```

When you copy and paste the previous command into the terminal with the `mysql` prefix, you should get an answer of 347.



```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| Chinook            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE Chinook;
Database changed
mysql> SELECT COUNT(*) FROM Album;
+----------+
| COUNT(*) |
+----------+
|      347 |
+----------+
1 row in set (0.00 sec)

mysql> 
```

Instantiate an LLM

This section guides you through the process of instantiating an LLM by using the watsonx.ai API. This section uses the `ibm/granite-3-2-8b-instruct` model. To find other foundational models that are available on watsonx.ai, refer to [Foundation model library].

Create the LLM

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

Open **llm_agent.py** in IDE

If file `llm_agent.py` does not create automatically, you can manually create it in this project.

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

```
# 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 = 'ibm/granite-3-2-8b-instruct'
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
)
granite_llm = WatsonxLLM(model = model)
print(granite_llm.invoke("What is the capital of Ontario?"))
```

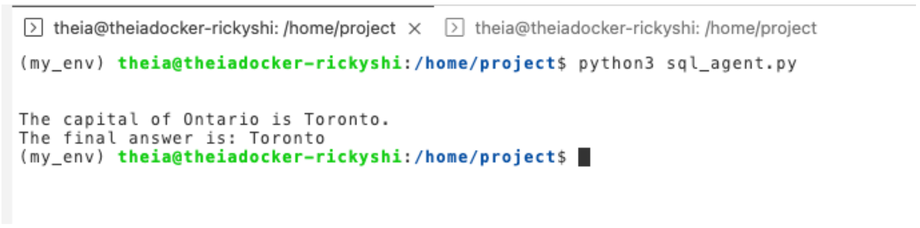
Note that in addition to creating the LLM object, the previous code includes the sample query 'What is the capital of Ontario?'. This query was included to test whether the model is being correctly loaded by the script.

Test the model

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

```
python3 llm_agent.py
```

If you were successful, you'll see a response in the terminal that's similar to the following response.


theia@theiadocker-rickyshi: /home/project
theia@theiadocker-rickyshi: /home/project

```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent.py


The capital of Ontario is Toronto.
The final answer is: Toronto
(my_env) theia@theiadocker-rickyshi:/home/project$
```

Great, the LLM is ready to be used!

Build the database connector

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

Open **sql_agent.py** in IDE

If file `sql_agent.py` does not create automatically, you can manually create it in this project.

The whole code will be provided later.

First, please copy and paste all the code in `llm_agent.py` to the `sql_agent.py` file.

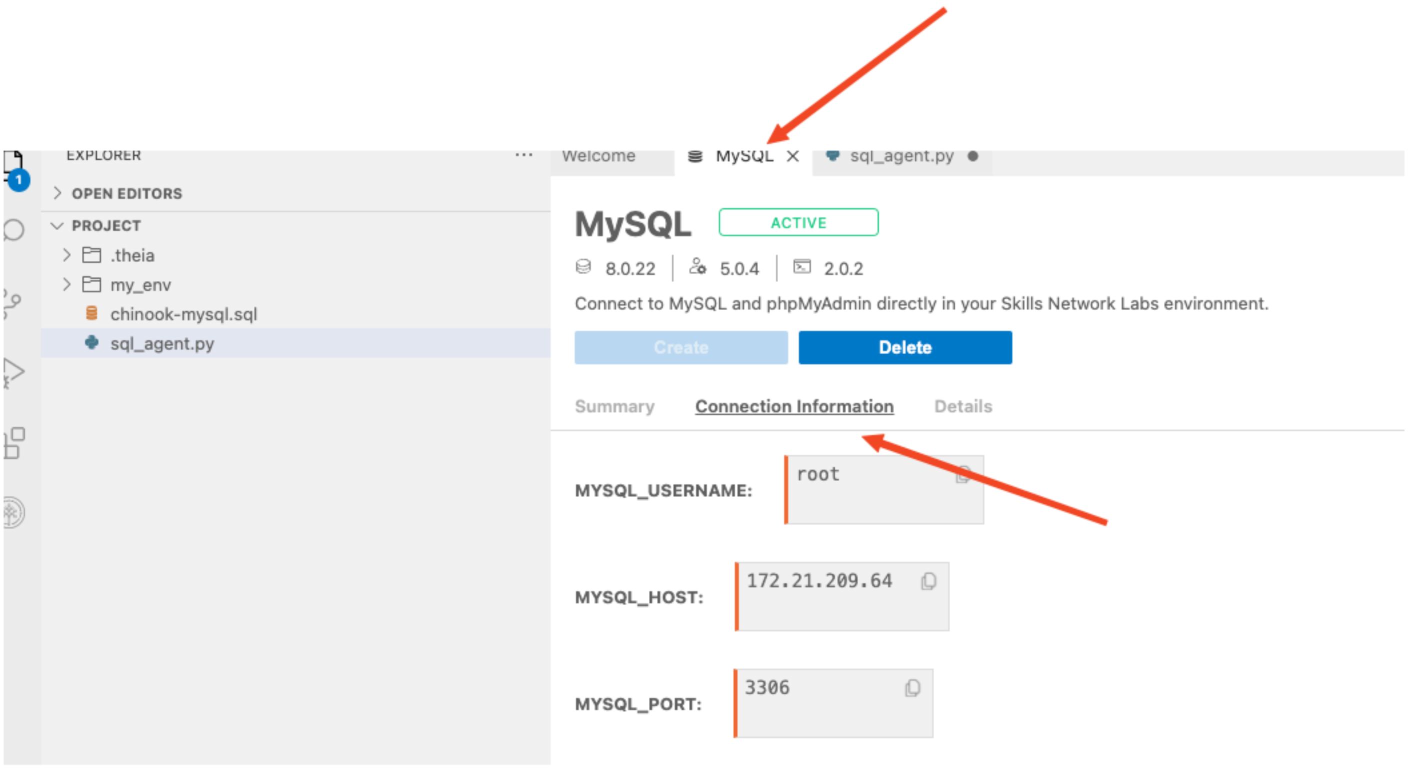
Building the database connector is simple. It requires just two lines of code:

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

You must define

- `mysql_username`
- `mysql_password`
- `mysql_host`
- `mysql_port`
- `database_name`.

To find the values of these parameters, go to the `MySQL` service. Then, click the `Connection Information` tab. Under this tab, you'll find almost all of the necessary parameter values, except for the `database_name`, which is the database you created earlier, `Chinook`.



MYSQL_USERNAME: root

MYSQL_HOST: 172.21.209.64

MYSQL_PORT: 3306

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

```
mysql_username = 'root'
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)
```

You should add the above code, to the `sql_agent.py` file.

Delete the following line from the `sql_agent.py` file.

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

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 to this point.

```
# 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 langchain.agents import AgentType
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 = 'ibm/granite-3-2-8b-instruct'
parameters = {
    GenParams.MAX_NEW_TOKENS: 1024,  # this controls the maximum number of tokens in the generated output
    GenParams.TEMPERATURE: 0.2, # this randomness or creativity of the model's responses
    GenParams.TOP_P: 0.95,
    GenParams.REPETITION_PENALTY: 1.2,
}
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
)
llm = WatsonxLLM(model = model)
mysql_username = 'root'  # Replace with your server connect information
mysql_password = 'TGbWkWxf2iPljYE3mYm8ilBj' # Replace with your server connect information
mysql_host = '172.21.155.85' # 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)
```

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.

`agent_executor = create_sql_agent(llm=llm, db=db, verbose=True, handle_parsing_errors=True, agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION)` is the code to create the SQL agent.

Parameters:

- `llm`: The LLM object that you created earlier.
- `db`: The database connector that you created earlier.
- `verbose`: Whether to print the verbose output.
- `handle_parsing_errors`: Whether to handle parsing errors.

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?"
)
```

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

```
# 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 langchain.agents import AgentType
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 = 'ibm/granite-3-2-8b-instruct'
parameters = {
    GenParams.MAX_NEW_TOKENS: 1024,  # this controls the maximum number of tokens in the generated output
    GenParams.TEMPERATURE: 0.2, # this randomness or creativity of the model's responses
    GenParams.TOP_P: 0.95,
    GenParams.REPETITION_PENALTY: 1.2,
}
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
)
llm = WatsonxLLM(model = model)
mysql_username = 'root'  # Replace with your server connect information
mysql_password = 'TGbWkWxf2iPljYE3mYm8ilBj' # Replace with your server connect information
mysql_host = '172.21.155.85' # 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)
agent_executor = create_sql_agent(llm=llm, db=db, verbose=True, handle_parsing_errors=True, agent_type=AgentType.ZERO_SHOT_REACT_DES
agent_executor.invoke(
    "How many Album are there in the database?"
)
```

Please copy and paste the previous code to the `sql_agent.py` file. Then, save the file and go to the terminal with the `my_env` prefix, and run the following command.

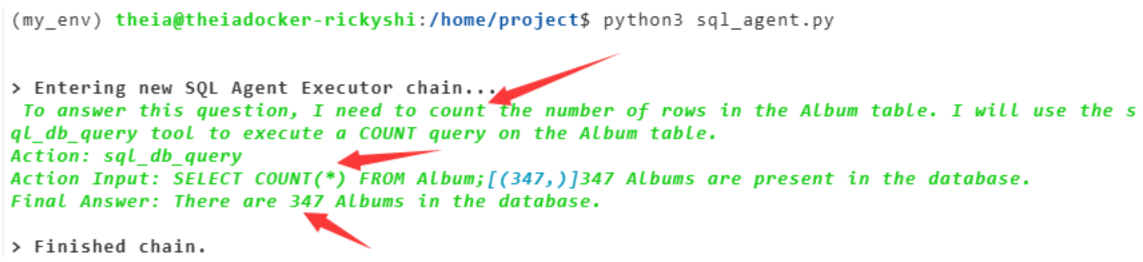
```
python3 sql_agent.py
```

If the agent runs successfully, you'll see a response similar to the following:

The Natural Language Query (NLQ) is:

```
How many Album are there in the database?
```

The response is:



```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent.py


> Entering new SQL Agent Executor chain...
 To answer this question, I need to count the number of rows in the Album table. I will use the s
ql_db_query tool to execute a COUNT query on the Album table.
Action: sql_db_query
Action Input: SELECT COUNT(*) FROM Album;[(347,)]347 Albums are present in the database.
Final Answer: There are 347 Albums in the database.

> Finished chain.
```

By setting `verbose=True` in the code, you get to see not just the `Final Answer`, but also the complete thought process - including all actions taken by the LLM and the actual SQL queries it generates to arrive at that answer. When running correctly, the agent will confirm there are 347 albums in the database, matching exactly what a direct SQL query would return.

NOTE: If you encounter any errors, please use `ctrl + C` in the terminal and run the code again.

Examples and Exercises

Please change the query in the `sql_agent.py` file to the following query and run it.

For example,

Change the query from:

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

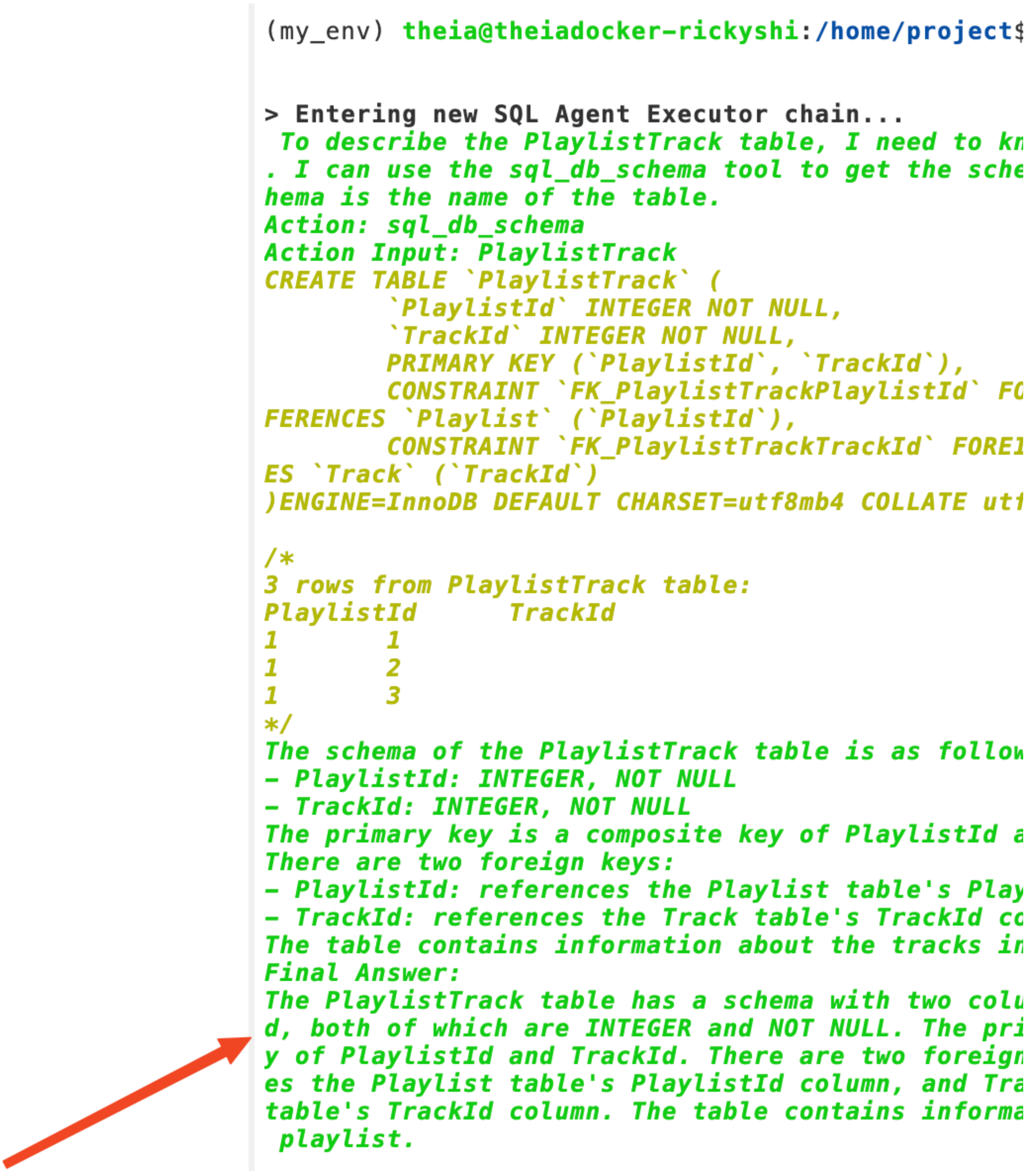
to:

```
agent_executor.invoke(
    "Describe the PlaylistTrack table"
)
```

Natural Language Query (NLQ):

```
Describe the PlaylistTrack table
```

Response:

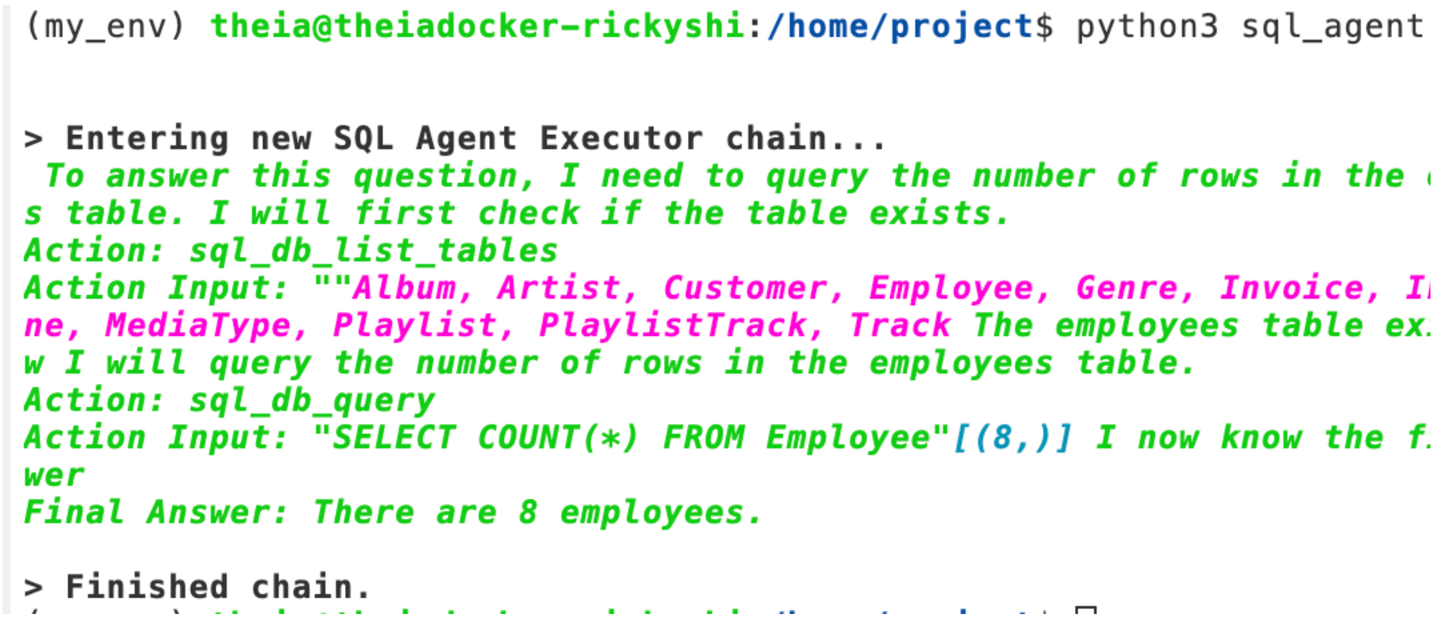


```
(my_env) theia@theiadocker-rickyshi:/home/project$


> Entering new SQL Agent Executor chain...
 To describe the PlaylistTrack table, I need to kn
. I can use the sql_db_schema tool to get the sche
hema is the name of the table.
Action: sql_db_schema
Action Input: PlaylistTrack
CREATE TABLE `PlaylistTrack` (
        `PlaylistId` INTEGER NOT NULL,
        `TrackId` INTEGER NOT NULL,
        PRIMARY KEY (`PlaylistId`, `TrackId`),
        CONSTRAINT `FK_PlaylistTrackPlaylistId` FO
FERENCES `Playlist` (`PlaylistId`),
        CONSTRAINT `FK_PlaylistTrackTrackId` FOREI
ES `Track` (`TrackId`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE utf

/*
3 rows from PlaylistTrack table:
PlaylistId      TrackId
1       1
1       2
1       3
*/
The schema of the PlaylistTrack table is as follow
- PlaylistId: INTEGER, NOT NULL
- TrackId: INTEGER, NOT NULL
The primary key is a composite key of PlaylistId a
There are two foreign keys:
- PlaylistId: references the Playlist table's Play
- TrackId: references the Track table's TrackId co
The table contains information about the tracks in
Final Answer:
The PlaylistTrack table has a schema with two colu
d, both of which are INTEGER and NOT NULL. The pri
y of PlaylistId and TrackId. There are two foreign
es the Playlist table's PlaylistId column, and Tra
table's TrackId column. The table contains informa
 playlist.
```

Natural Language Query (NLQ):

```
How many employees are there
```

Response:

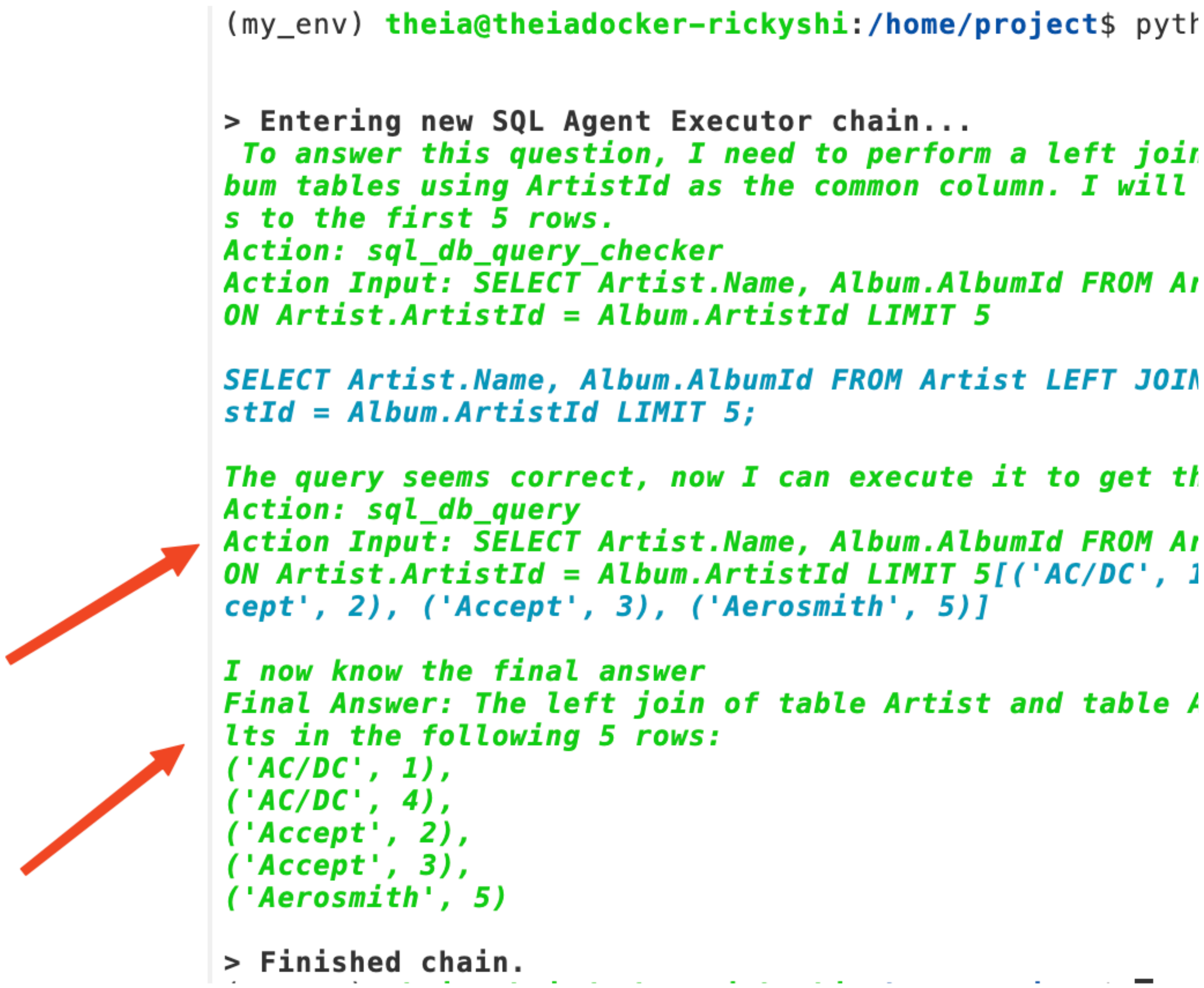


```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent


> Entering new SQL Agent Executor chain...
 To answer this question, I need to query the number of rows in the 
s table. I will first check if the table exists.
Action: sql_db_list_tables
Action Input: ""Album, Artist, Customer, Employee, Genre, Invoice, I
ne, MediaType, Playlist, PlaylistTrack, Track The employees table ex
w I will query the number of rows in the employees table.
Action: sql_db_query
Action Input: "SELECT COUNT(*) FROM Employee"[(8,)] I now know the f
wer
Final Answer: There are 8 employees.

> Finished chain.
```

Natural Language Query (NLQ):

```
Can you left join table Artist and table Album by ArtistId? Please show me 5 Name and AlbumId in the joint table.
```

Response:

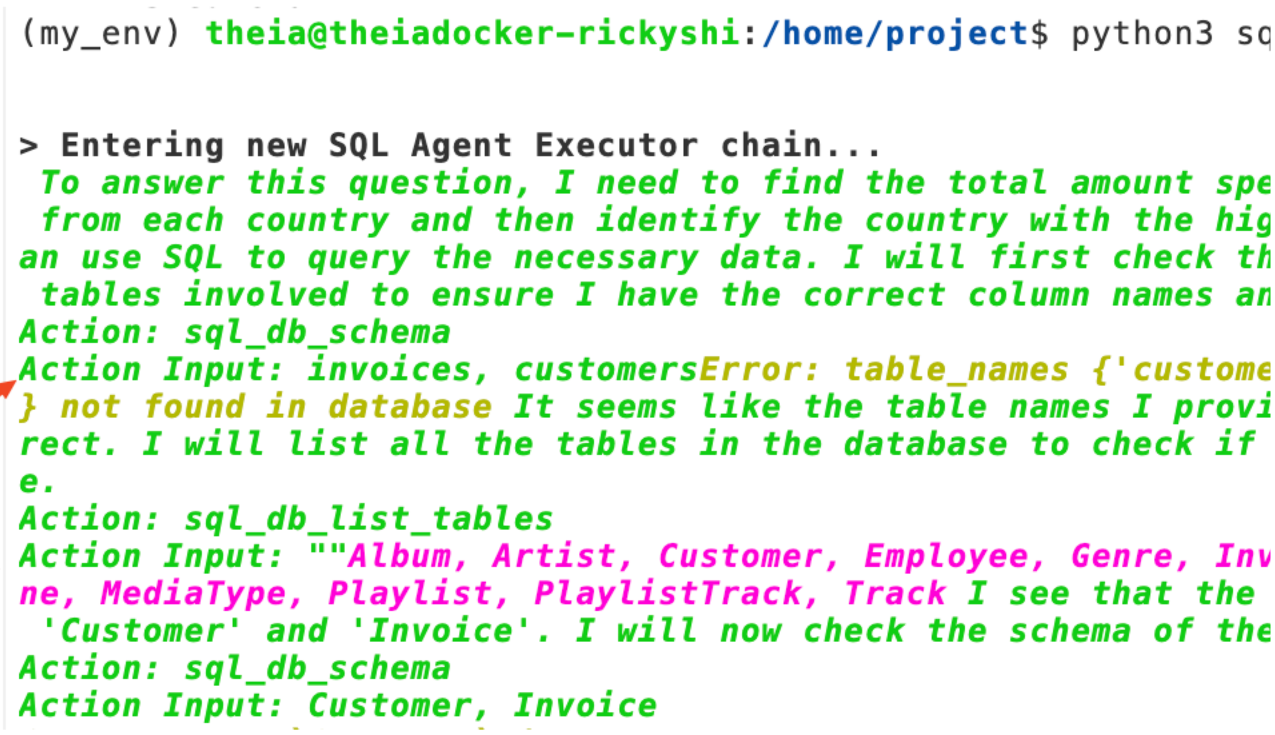


```
(my_env) theia@theiadocker-rickyshi:/home/project$ pytl


> Entering new SQL Agent Executor chain...
 To answer this question, I need to perform a left joir
bum tables using ArtistId as the common column. I will
s to the first 5 rows.
Action: sql_db_query_checker
Action Input: SELECT Artist.Name, Album.AlbumId FROM Ar
ON Artist.ArtistId = Album.ArtistId LIMIT 5

SELECT Artist.Name, Album.AlbumId FROM Artist LEFT JOIN
stId = Album.ArtistId LIMIT 5;

The query seems correct, now I can execute it to get th
Action: sql_db_query
Action Input: SELECT Artist.Name, Album.AlbumId FROM Ar
ON Artist.ArtistId = Album.ArtistId LIMIT 5[('AC/DC', 1
cept', 2), ('Accept', 3), ('Aerosmith', 5)]

I now know the final answer
Final Answer: The left join of table Artist and table A
lts in the following 5 rows:
('AC/DC', 1),
('AC/DC', 4),
('Accept', 2),
('Accept', 3),
('Aerosmith', 5)

> Finished chain.
```

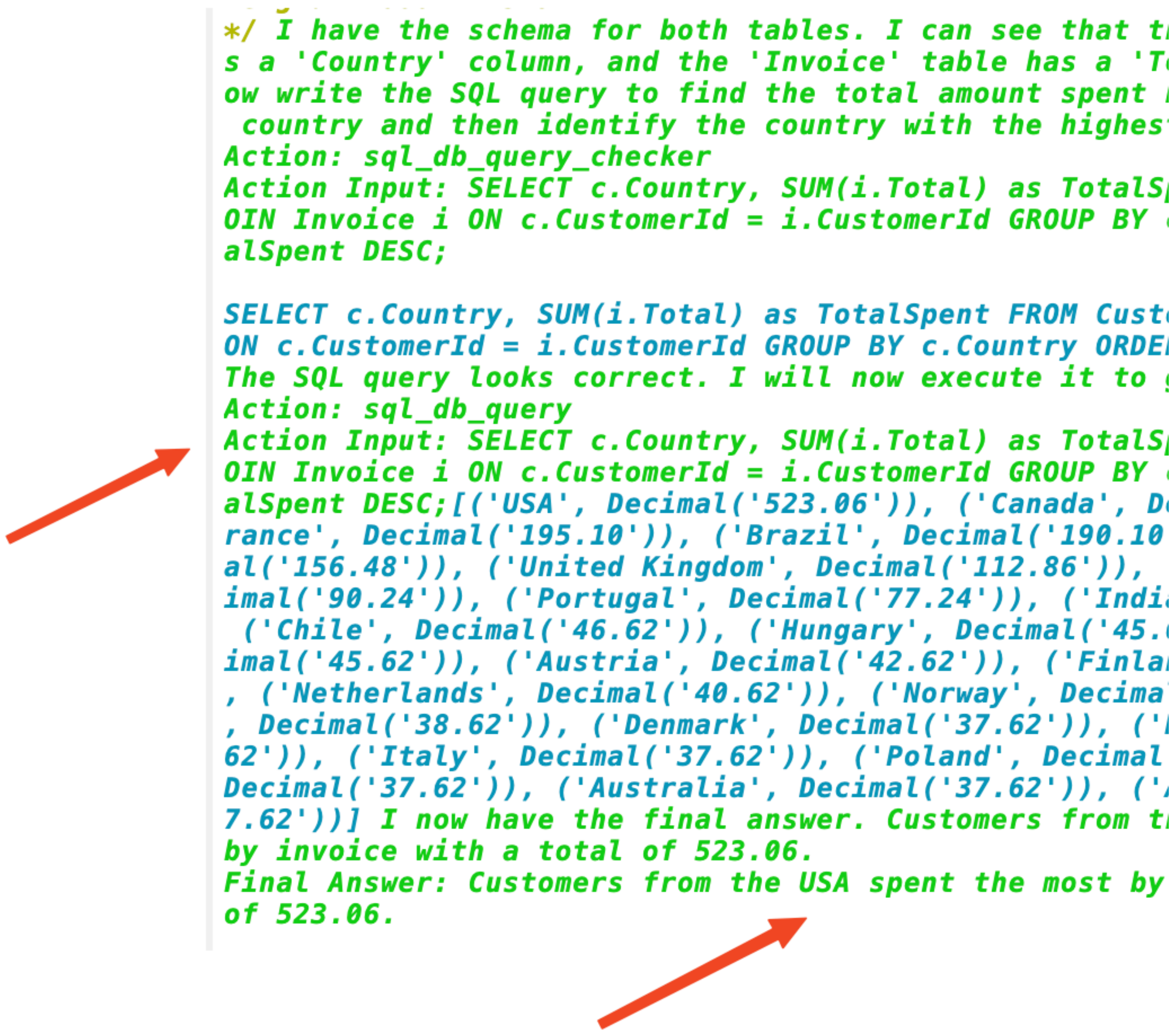
Natural Language Query (NLQ):

```
Which country's customers spent the most by invoice?
```

Response:



```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sq


> Entering new SQL Agent Executor chain...
 To answer this question, I need to find the total amount spe
 from each country and then identify the country with the hig
an use SQL to query the necessary data. I will first check th
 tables involved to ensure I have the correct column names an
Action: sql_db_schema
Action Input: invoices, customersError: table_names {'custome
} not found in database It seems like the table names I provi
rect. I will list all the tables in the database to check if
e.
Action: sql_db_list_tables
Action Input: ""Album, Artist, Customer, Employee, Genre, Inv
ne, MediaType, Playlist, PlaylistTrack, Track I see that the
 'Customer' and 'Invoice'. I will now check the schema of the
Action: sql_db_schema
Action Input: Customer, Invoice
```



```
*/ I have the schema for both tables. I can see that t
s a 'Country' column, and the 'Invoice' table has a 'T
ow write the SQL query to find the total amount spent 
 country and then identify the country with the highes
Action: sql_db_query_checker
Action Input: SELECT c.Country, SUM(i.Total) as TotalS
OIN Invoice i ON c.CustomerId = i.CustomerId GROUP BY 
alSpent DESC;

SELECT c.Country, SUM(i.Total) as TotalSpent FROM Cust
ON c.CustomerId = i.CustomerId GROUP BY c.Country ORDE
The SQL query looks correct. I will now execute it to 
Action: sql_db_query
Action Input: SELECT c.Country, SUM(i.Total) as TotalS
OIN Invoice i ON c.CustomerId = i.CustomerId GROUP BY 
alSpent DESC;[('USA', Decimal('523.06')), ('Canada', D
rance', Decimal('195.10')), ('Brazil', Decimal('190.10
al('156.48')), ('United Kingdom', Decimal('112.86')), 
imal('90.24')), ('Portugal', Decimal('77.24')), ('Indi
 ('Chile', Decimal('46.62')), ('Hungary', Decimal('45.
imal('45.62')), ('Austria', Decimal('42.62')), ('Finla
, ('Netherlands', Decimal('40.62')), ('Norway', Decima
, Decimal('38.62')), ('Denmark', Decimal('37.62')), ('
62')), ('Italy', Decimal('37.62')), ('Poland', Decimal
Decimal('37.62')), ('Australia', Decimal('37.62')), ('
7.62'))] I now have the final answer. Customers from t
by invoice with a total of 523.06.
Final Answer: Customers from the USA spent the most by
of 523.06.
```

Use the command line to run Natural Language Query (NLQ)

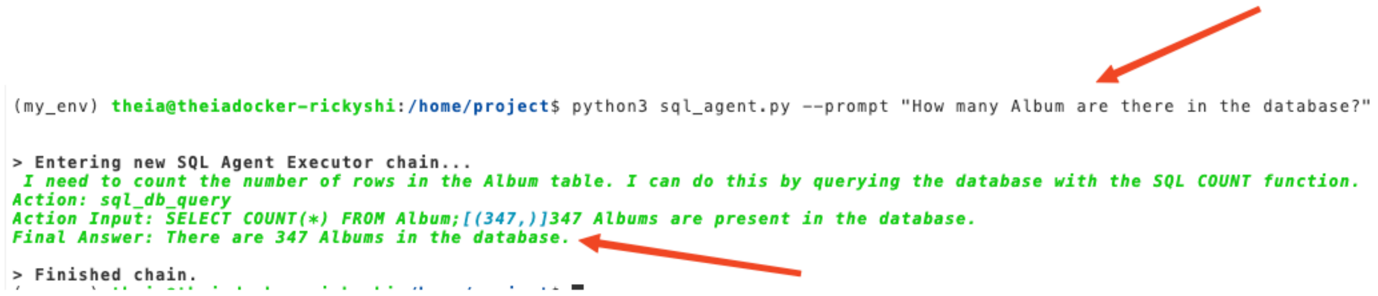
Copy and paste the following code to the `sql_agent.py` file.

NOTE: remember to replace the `mysql_username`, `mysql_password`, `mysql_host`, `mysql_port`, and `database_name` with your own values.

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')
from ibm_watsonx_ai.foundation_models import ModelInference
from langchain.agents import AgentType
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
import argparse
# Set up argument parser
parser = argparse.ArgumentParser()
parser.add_argument("--prompt", type=str, help="The prompt to send to the SQL agent")
args = parser.parse_args()
model_id = 'ibm/granite-3-2-8b-instruct'
parameters = {
    GenParams.MAX_NEW_TOKENS: 1024,  # this controls the maximum number of tokens in the generated output
    GenParams.TEMPERATURE: 0.2, # this randomness or creativity of the model's responses
    GenParams.TOP_P: 0.95,
    GenParams.REPETITION_PENALTY: 1.2,
}
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
)
llm = WatsonxLLM(model = model)
mysql_username = 'root'  # Replace with your server connect information
mysql_password = 'nVEOA1iQqBOz4cftLGdhXuTu' # Replace with your server connect information
mysql_host = '172.21.47.178' # 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)
agent_executor = create_sql_agent(llm=llm, db=db, verbose=True, handle_parsing_errors=True, agent_type=AgentType.ZERO_SHOT_REACT_DES
# Use the prompt from command line argument
if args.prompt:
    agent_executor.invoke(args.prompt)
else:
    print("Please provide a prompt using --prompt argument")
```

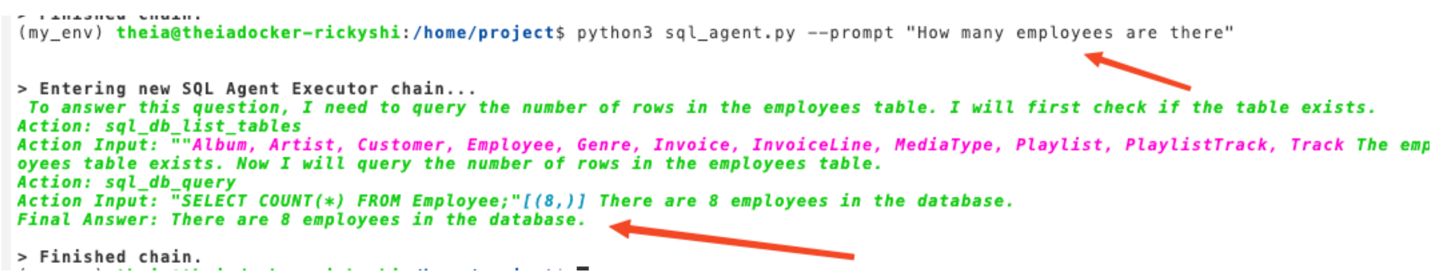
To run the code, you can use the following command in the terminal with the `my_env` prefix.

```
python3 sql_agent.py --prompt "How many Album are there in the database?"
```



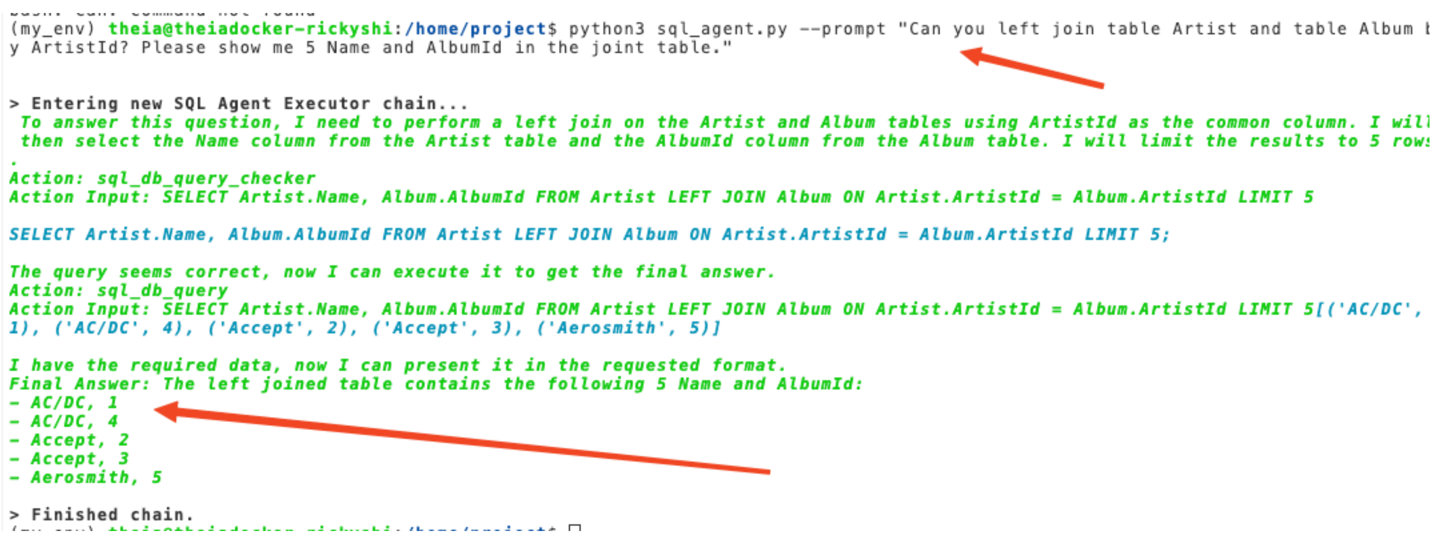
```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent.py --prompt "How many Album are there in the database?"

> Entering new SQL Agent Executor chain...
 I need to count the number of rows in the Album table. I can do this by querying the database with the SQL COUNT function.
Action: sql_db_query
Action Input: SELECT COUNT(*) FROM Album;[(347,)]347 Albums are present in the database.
Final Answer: There are 347 Albums in the database.

> Finished chain.
```

```
python3 sql_agent.py --prompt "How many employees are there"
```



```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent.py --prompt "How many employees are there"

> Entering new SQL Agent Executor chain...
 To answer this question, I need to query the number of rows in the employees table. I will first check if the table exists.
Action: sql_db_list_tables
Action Input: ""Album, Artist, Customer, Employee, Genre, Invoice, InvoiceLine, MediaType, Playlist, PlaylistTrack, Track The emp
oyees table exists. Now I will query the number of rows in the employees table.
Action: sql_db_query
Action Input: "SELECT COUNT(*) FROM Employee;"[(8,)] There are 8 employees in the database.
Final Answer: There are 8 employees in the database.

> Finished chain.
```

```
python3 sql_agent.py --prompt "Can you left join table Artist and table Album by ArtistId? Please show me 5 Name and AlbumId in the
```



```
(my_env) theia@theiadocker-rickyshi:/home/project$ python3 sql_agent.py --prompt "Can you left join table Artist and table Album b
y ArtistId? Please show me 5 Name and AlbumId in the joint table."


> Entering new SQL Agent Executor chain...
 To answer this question, I need to perform a left join on the Artist and Album tables using ArtistId as the common column. I will
 then select the Name column from the Artist table and the AlbumId column from the Album table. I will limit the results to 5 rows
.
Action: sql_db_query_checker
Action Input: SELECT Artist.Name, Album.AlbumId FROM Artist LEFT JOIN Album ON Artist.ArtistId = Album.ArtistId LIMIT 5

SELECT Artist.Name, Album.AlbumId FROM Artist LEFT JOIN Album ON Artist.ArtistId = Album.ArtistId LIMIT 5;

The query seems correct, now I can execute it to get the final answer.
Action: sql_db_query
Action Input: SELECT Artist.Name, Album.AlbumId FROM Artist LEFT JOIN Album ON Artist.ArtistId = Album.ArtistId LIMIT 5[('AC/DC',
1), ('AC/DC', 4), ('Accept', 2), ('Accept', 3), ('Aerosmith', 5)]

I have the required data, now I can present it in the requested format.
Final Answer: The left joined table contains the following 5 Name and AlbumId:
- AC/DC, 1
- AC/DC, 4
- Accept, 2
- Accept, 3
- Aerosmith, 5

> Finished chain.
```

Please try more examples by yourself:

```
python3 sql_agent.py --prompt "Describe the PlaylistTrack table"
```

```
python3 sql_agent.py --prompt "Which country's customers spent the most by invoice?"
```

Congratulations, you have finished the project!

Author(s)

[Ricky Shi](#)
Ricky Shi is a Data Scientist at IBM.

Contributors

[Karan Goswami](#)
[Kunal Makwana](#)
[Wojciech "Victor" Fulmyk](#)