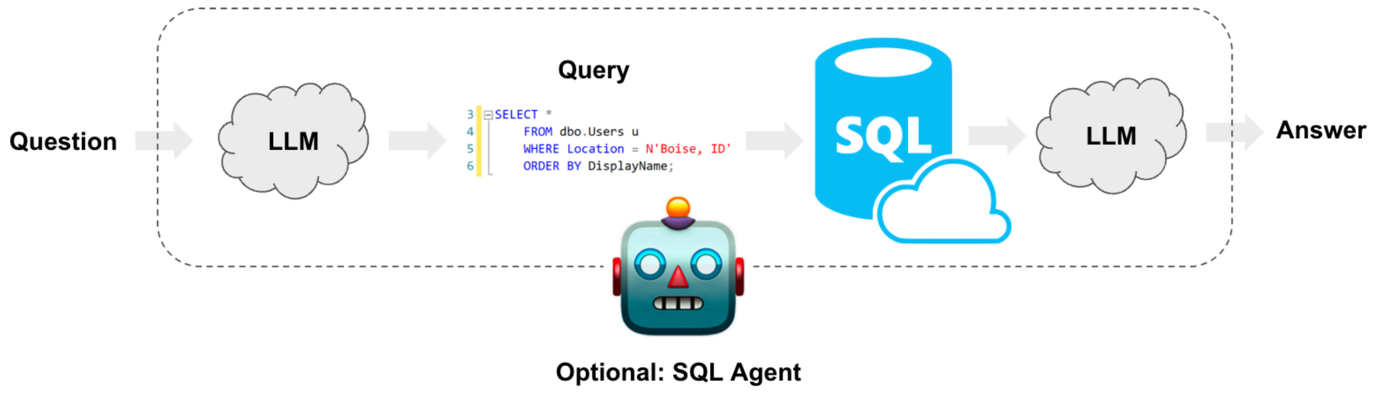


```
SELECT *
    FROM dbo.Users u
    WHERE Location = N'Boise, ID'
    ORDER BY DisplayName;
```

Readme

首先需要在`.env`文件中填写API_KEY

```
ARK_API_KEY=""
ARK_API_URL=""
MODEL_NAME=""
LANGCHAIN_TRACING_V2="true"
LANGCHAIN_API_KEY=""
PANDASAI_API_KEY=""
```

这里需要填写langchain、pandasai以及llm的API，除了LLM的API之外，其他两个的API申请都是免费的。

创建数据库

然后需要创建数据库，本文中有两个数据库netflows(自己随机生成的数据库)和employees(mysql经典数据库)

netflows

```
create database netflows;
use netflows;
CREATE TABLE packets (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- 自动递增的主键
    src_ip VARCHAR(45) NOT NULL,        -- 源IP地址，不能为空
    dst_ip VARCHAR(45) NOT NULL,        -- 目的IP地址，不能为空
    src_port INT DEFAULT NULL,          -- 源端口，可以为空
    dst_port INT DEFAULT NULL,          -- 目的端口，可以为空
    protocol VARCHAR(10) DEFAULT NULL,  -- 协议（例如 'TCP', 'UDP'），可以为空
    timestamp TIMESTAMP(6) NOT NULL,    -- 数据包时间戳，精确到微秒，不能为空
    size INT DEFAULT NULL,              -- 数据包大小（字节），可以为空
    INDEX idx_timestamp (timestamp)    -- 为 timestamp 字段创建索引，加速查询排序
);
INSERT INTO packets (src_ip, dst_ip, src_port, dst_port, protocol, timestamp, size)
VALUES
('192.168.1.1', '192.168.1.2', 12345, 80, 'TCP', '2024-11-30 10:00:00', 512),
('192.168.1.3', '192.168.1.4', 54321, 443, 'UDP', '2024-11-30 10:05:00', 256),
('192.168.1.5', '192.168.1.6', 23456, 22, 'TCP', '2024-11-30 10:10:00', 1024),
('192.168.1.7', '192.168.1.8', 34567, 8080, 'TCP', '2024-11-30 10:15:00', 128),
('192.168.1.9', '192.168.1.10', 45678, 53, 'UDP', '2024-11-30 10:20:00', 512),
('192.168.1.11', '192.168.1.12', 56789, 443, 'TCP', '2024-11-30 10:25:00', 1024),
('192.168.1.13', '192.168.1.14', 67890, 80, 'UDP', '2024-11-30 10:30:00', 256),
('192.168.1.15', '192.168.1.16', 78901, 25, 'TCP', '2024-11-30 10:35:00', 128),
('192.168.1.17', '192.168.1.18', 89012, 443, 'UDP', '2024-11-30 10:40:00', 512),
('192.168.1.19', '192.168.1.20', 90123, 21, 'TCP', '2024-11-30 10:45:00', 1024),
('192.168.1.21', '192.168.1.22', 12345, 110, 'UDP', '2024-11-30 10:50:00', 256),
('192.168.1.23', '192.168.1.24', 23456, 25, 'TCP', '2024-11-30 10:55:00', 512),
('192.168.1.25', '192.168.1.26', 34567, 443, 'UDP', '2024-11-30 11:00:00', 128),
('192.168.1.27', '192.168.1.28', 45678, 80, 'TCP', '2024-11-30 11:05:00', 1024),
('192.168.1.29', '192.168.1.30', 56789, 22, 'UDP', '2024-11-30 11:10:00', 256),
('192.168.1.31', '192.168.1.32', 67890, 8080, 'TCP', '2024-11-30 11:15:00', 128),
('192.168.1.33', '192.168.1.34', 78901, 53, 'UDP', '2024-11-30 11:20:00', 512),
('192.168.1.35', '192.168.1.36', 89012, 443, 'TCP', '2024-11-30 11:25:00', 1024),
('192.168.1.37', '192.168.1.38', 90123, 21, 'UDP', '2024-11-30 11:30:00', 256),
('192.168.1.39', '192.168.1.40', 12345, 25, 'TCP', '2024-11-30 11:35:00', 128),
('192.168.1.41', '192.168.1.42', 23456, 443, 'UDP', '2024-11-30 11:40:00', 512),
('192.168.1.43', '192.168.1.44', 34567, 443, 'TCP', '2024-11-30 11:45:00', 1024),
('192.168.1.45', '192.168.1.46', 45678, 8080, 'UDP', '2024-11-30 11:50:00', 256),
('192.168.1.47', '192.168.1.48', 56789, 80, 'TCP', '2024-11-30 11:55:00', 128),
('192.168.1.49', '192.168.1.50', 67890, 443, 'UDP', '2024-11-30 12:00:00', 512),
('192.168.1.51', '192.168.1.52', 78901, 110, 'TCP', '2024-11-30 12:05:00', 1024),
('192.168.1.53', '192.168.1.54', 89012, 443, 'UDP', '2024-11-30 12:10:00', 256),
('192.168.1.55', '192.168.1.56', 90123, 25, 'TCP', '2024-11-30 12:15:00', 128),
('192.168.1.57', '192.168.1.58', 12345, 53, 'UDP', '2024-11-30 12:20:00', 512),
('192.168.1.59', '192.168.1.60', 23456, 22, 'TCP', '2024-11-30 12:25:00', 1024),
('192.168.1.61', '192.168.1.62', 34567, 443, 'UDP', '2024-11-30 12:30:00', 256),
('192.168.1.63', '192.168.1.64', 45678, 80, 'TCP', '2024-11-30 12:35:00', 128),
('192.168.1.65', '192.168.1.66', 56789, 8080, 'UDP', '2024-11-30 12:40:00', 512),
('192.168.1.67', '192.168.1.68', 67890, 443, 'TCP', '2024-11-30 12:45:00', 1024),
('192.168.1.69', '192.168.1.70', 78901, 25, 'UDP', '2024-11-30 12:50:00', 256),
('192.168.1.71', '192.168.1.72', 89012, 21, 'TCP', '2024-11-30 12:55:00', 128),
('192.168.1.73', '192.168.1.74', 90123, 443, 'UDP', '2024-11-30 13:00:00', 512),
('192.168.1.75', '192.168.1.76', 12345, 8080, 'TCP', '2024-11-30 13:05:00', 1024),
('192.168.1.77', '192.168.1.78', 23456, 80, 'UDP', '2024-11-30 13:10:00', 256),
('192.168.1.79', '192.168.1.80', 34567, 25, 'TCP', '2024-11-30 13:15:00', 128),
('192.168.1.81', '192.168.1.82', 45678, 443, 'UDP', '2024-11-30 13:20:00', 512),
('192.168.1.83', '192.168.1.84', 56789, 8080, 'TCP', '2024-11-30 13:25:00', 1024),
('192.168.1.85', '192.168.1.86', 67890, 21, 'UDP', '2024-11-30 13:30:00', 256);
```

employees

```
wget 'https://codeload.github.com/datacharmer/test_db/zip/master' -O test_db-master.zip
unzip test_db-master.zip
cd test_db-master
mysql -u root -p88888888 < employees.sql
```

Agent运行

实现了两个Agent：

- llmAgent.py: 用langchain实现，包括画图部分也是用的langchain

- llmAgent_pandasai.py: 用langchain+pandasai实现，生成sql语句后将查询结果df交给pandasai进行处理

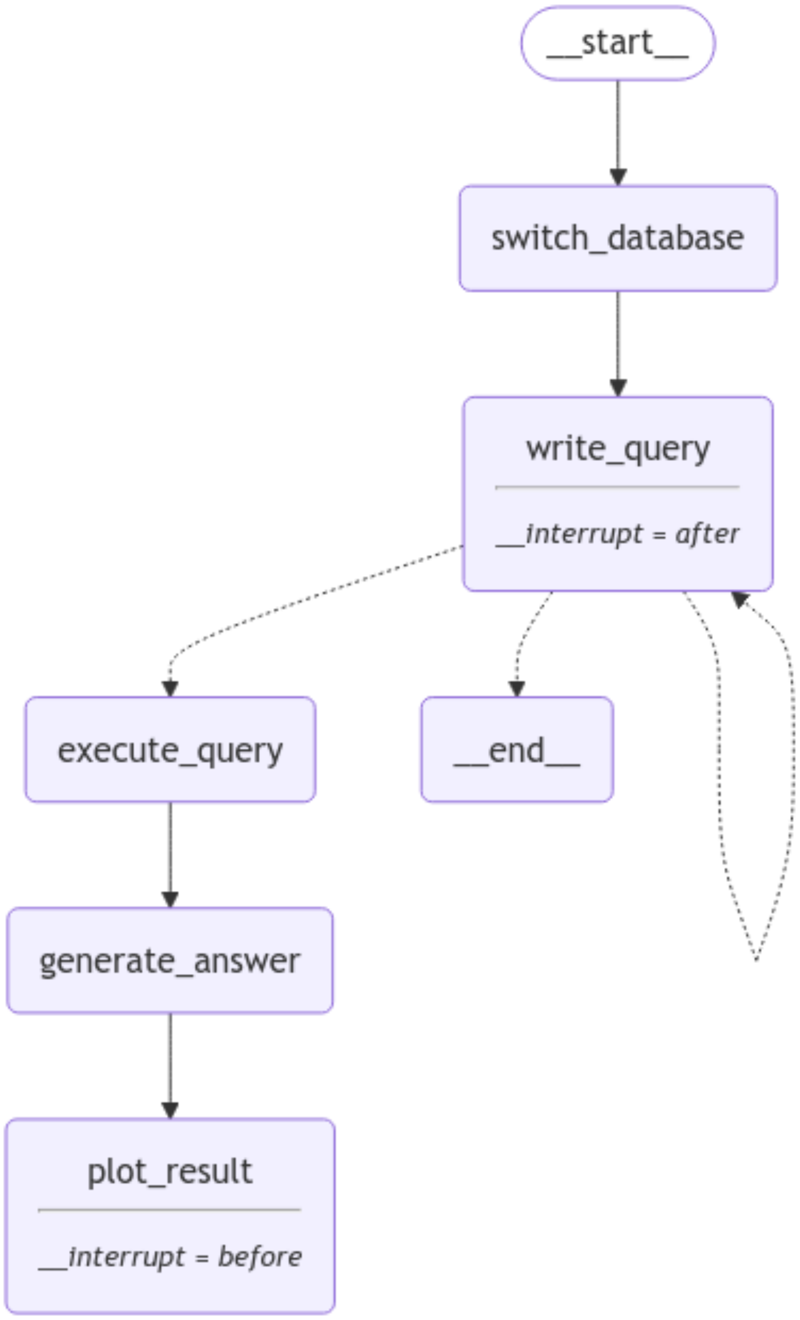
运行

```
python llmAgent.py
python llmAgent_pandasai.py
```

两个的流程都一样，先输入需要查询的数据内容，llm返回sql指令后，用户检查一遍，输入yes则执行查询，输入no则会重新生成sql指令，输入exit则会退出程序。输出查询结果后，用户可以输入绘图要求，Agent会自动生成图片，llmAgent.py会生成在./plot.png，llmAgent_pandasai.py则会生成在./exports/charts下。

实验报告

基本的流程如下



该图由langchain的函数`display(Image(app.get_graph().draw_mermaid_png()))`自动生成.

先输入input，然后llm根据input首先进行switch_database（考虑到多数据源），然后write_query生成sql查询，用户检查查询之后，根据输入yes、no和end分别进行执行、重写sql以及退出的操作，执行完sql查询后会由llm总结答案并展示（pandasai没有实现这个，纯llm版本实现了），然后用户输入plot需求进行绘图。

Langchain

Langchain 是一个开源框架，它允许开发人员将像 GPT-4 这样的大型语言模型与外部的计算和数据源结合起来。

用stategraph串联起整个agent的工作流程，其构建如下

```
workflow = StateGraph(State)
workflow.add_node("switch_database", switch_database)
workflow.add_node("write_query", write_query)
workflow.add_node("execute_query", execute_query)
workflow.add_node("generate_answer", generate_answer)
workflow.add_node("plot_result", plot_result)
workflow.add_edge(START, "switch_database")
workflow.add_edge("switch_database", "write_query")
workflow.add_conditional_edges("write_query", should_continue)
workflow.add_edge("execute_query", "generate_answer")
workflow.add_edge("generate_answer", "plot_result")
memory = MemorySaver()
app = workflow.compile(checkpointer=memory, interrupt_after=["write_query"],
interrupt_before=["plot_result"])
config = {"configurable": {"thread_id": "1"}}
display(Image(app.get_graph().draw_mermaid_png()))
```

写完函数之后就能像图一样构建Agent，运行逻辑也可以自己写

```
    question = input(">")
    for step in app.stream(
        {"question": question},
        config,
        stream_mode="updates",
    ):
        print(step)
    user_approval = "yes"
    while user_approval != "exit":
        try:
            user_approval = input("Do you want to go to execute query? (yes/no/exit): ")
        except Exception:
            user_approval = "no"

        if user_approval.lower() == "yes":
            app.update_state(config, {"res": "yes"})
        elif user_approval.lower() == "no":
            app.update_state(config, {"res": "no"})
        else:
            app.update_state(config, {"res": "exit"})
            print("Agent Exit by User.")
            exit()
        for step in app.stream(None, config, stream_mode="updates"):
            print(step)
        if user_approval.lower() == "yes":
            break
    try:
        plot_request = input("What do you want to plot? (e.g. histogram, scatter plot):
")
    except Exception:
        plot_request = "no"
    if plot_request == 'no':
        exit()
    else:
        app.update_state(config, {"plot_request": plot_request})
    for step in app.stream(None, config, stream_mode="updates"):
        print(step)
```

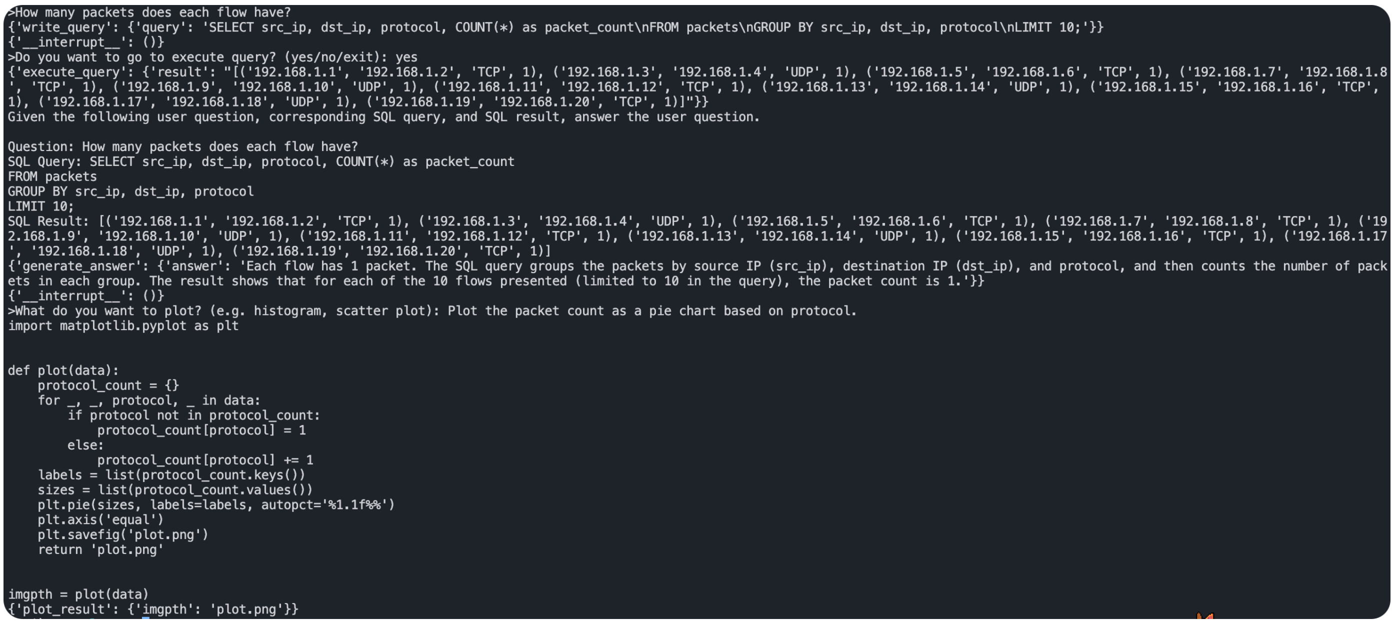
本项目中生成sql的prompt采用的langchain库中自带的，测试下来性能较好，能完成多表联合查询

```
================================•[1m System Message •[0m================================

Given an input question, create a syntactically correct {dialect} query to run to help
find the answer. Unless the user specifies in his question a specific number of examples
they wish to obtain, always limit your query to at most {topk} results. You can order the
results by a relevant column to return the most interesting examples in the database.

Never query for all the columns from a specific table, only ask for a the few relevant
columns given the question.

Pay attention to use only the column names that you can see in the schema description. Be
careful to not query for columns that do not exist. Also, pay attention to which column
is in which table.

Only use the following tables:
{table_info}

Question: {input}
```

实验结果

llmAgent.py

一轮对话如下



```
>How many packets does each flow have?
{'write_query': {'query': 'SELECT src_ip, dst_ip, protocol, COUNT(*) as packet_count\nFROM packets\nGROUP BY src_ip, dst_ip, protocol\nLIMIT 10;'}}
{'__interrupt__': ()}
>Do you want to go to execute query? (yes/no/exit): yes
{'execute_query': {'result': "[('192.168.1.1', '192.168.1.2', 'TCP', 1), ('192.168.1.3', '192.168.1.4', 'UDP', 1), ('192.168.1.5', '192.168.1.6', 'TCP', 1), ('192.168.1.7', '192.168.1.8
', 'TCP', 1), ('192.168.1.9', '192.168.1.10', 'UDP', 1), ('192.168.1.11', '192.168.1.12', 'TCP', 1), ('192.168.1.13', '192.168.1.14', 'UDP', 1), ('192.168.1.15', '192.168.1.16', 'TCP',
1), ('192.168.1.17', '192.168.1.18', 'UDP', 1), ('192.168.1.19', '192.168.1.20', 'TCP', 1)]"}}
Given the following user question, corresponding SQL query, and SQL result, answer the user question.

Question: How many packets does each flow have?
SQL Query: SELECT src_ip, dst_ip, protocol, COUNT(*) as packet_count
FROM packets
GROUP BY src_ip, dst_ip, protocol
LIMIT 10;
SQL Result: [('192.168.1.1', '192.168.1.2', 'TCP', 1), ('192.168.1.3', '192.168.1.4', 'UDP', 1), ('192.168.1.5', '192.168.1.6', 'TCP', 1), ('192.168.1.7', '192.168.1.8', 'TCP', 1), ('19
2.168.1.9', '192.168.1.10', 'UDP', 1), ('192.168.1.11', '192.168.1.12', 'TCP', 1), ('192.168.1.13', '192.168.1.14', 'UDP', 1), ('192.168.1.15', '192.168.1.16', 'TCP', 1), ('192.168.1.17
', '192.168.1.18', 'UDP', 1), ('192.168.1.19', '192.168.1.20', 'TCP', 1)]
{'generate_answer': {'answer': 'Each flow has 1 packet. The SQL query groups the packets by source IP (src_ip), destination IP (dst_ip), and protocol, and then counts the number of pack
ets in each group. The result shows that for each of the 10 flows presented (limited to 10 in the query), the packet count is 1.'}}
{'__interrupt__': ()}
>What do you want to plot? (e.g. histogram, scatter plot): Plot the packet count as a pie chart based on protocol.
import matplotlib.pyplot as plt


def plot(data):
    protocol_count = {}
    for _, _, protocol, _ in data:
        if protocol not in protocol_count:
            protocol_count[protocol] = 1
        else:
            protocol_count[protocol] += 1
    labels = list(protocol_count.keys())
    sizes = list(protocol_count.values())
    plt.pie(sizes, labels=labels, autopct='%1.1f%%')
    plt.axis('equal')
    plt.savefig('plot.png')
    return 'plot.png'


imgpth = plot(data)
{'plot_result': {'imgpth': 'plot.png'}}
```

可以看到输入为`How many packets does each flow have?`

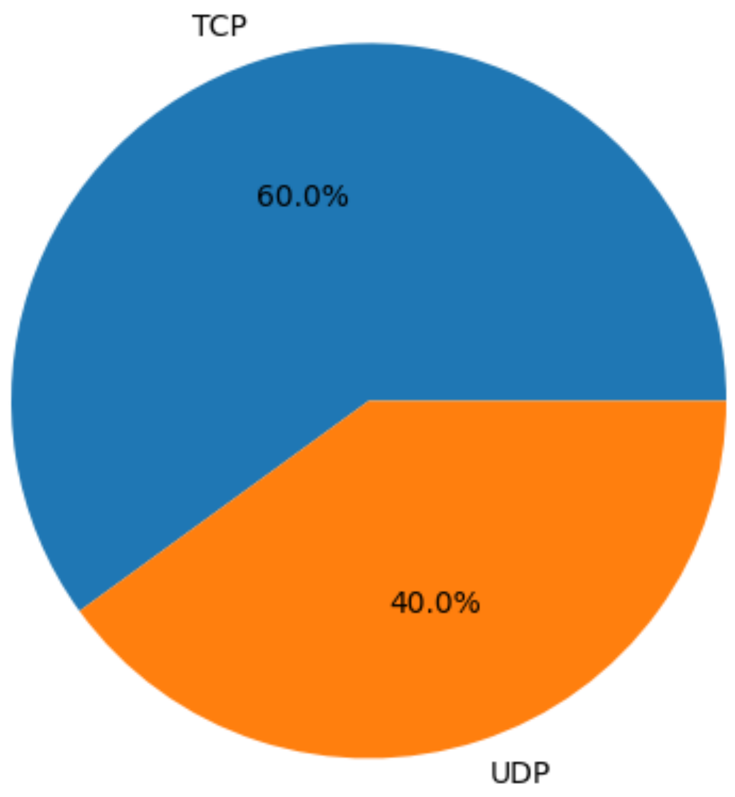
agent通过`write_query`函数生成prompt后发送给LLM，得到SQL查询语句

```
SELECT src_ip, dst_ip, protocol, COUNT(*) as packet_count
FROM packets
GROUP BY src_ip, dst_ip, protocol
LIMIT 10;
```

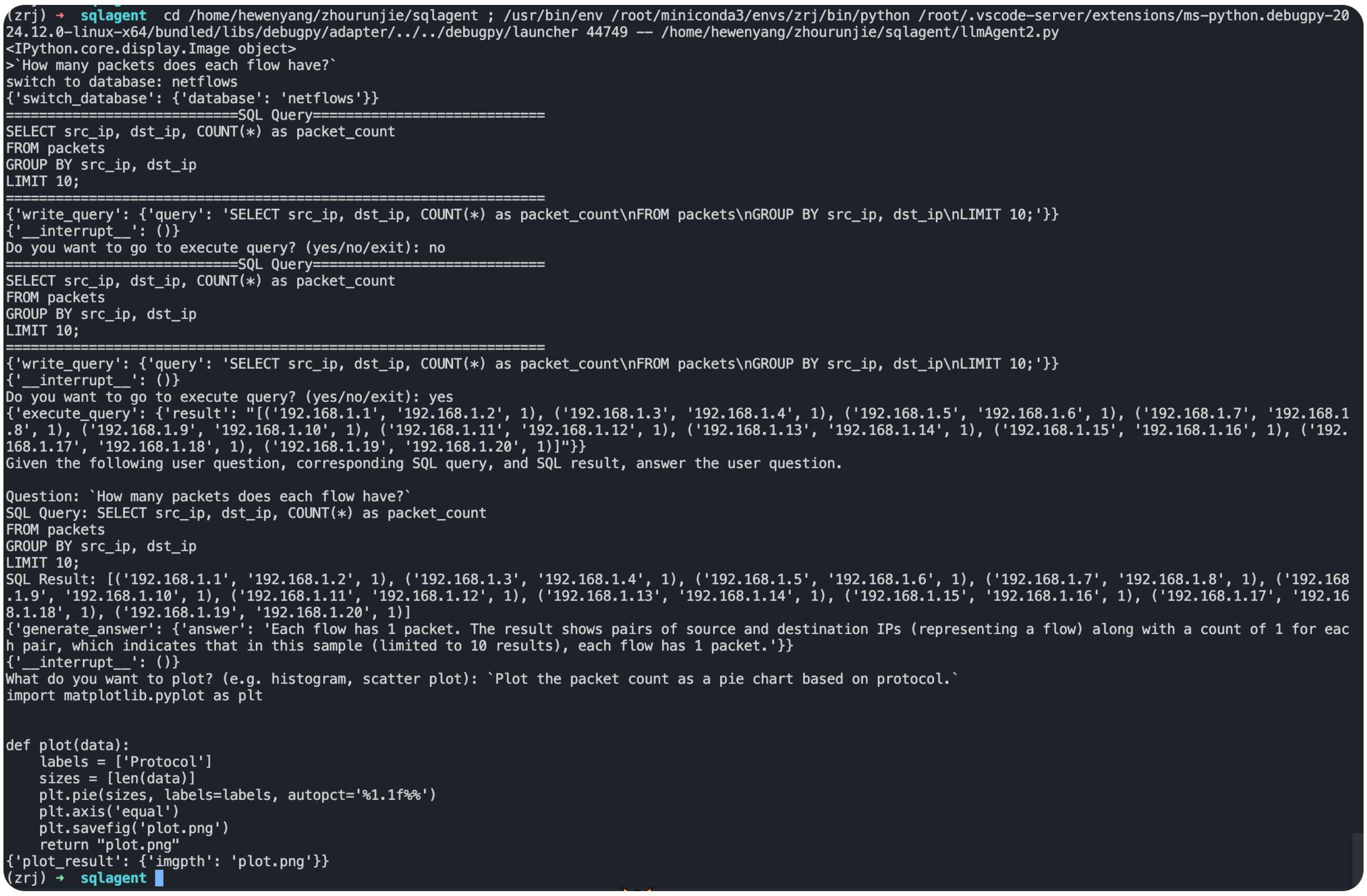
有了这个回答后，agent会询问是否要执行该SQL语句（让用户可以检查该语句，如果输入`yes`则会执行该SQL查询，如果输入`no`则会转到`write_query`重新生成一遍SQL，如果`exit`则退出该agent，这部分操作是通过langchain的`conditional_edge`实现的）

接下来通过`execute_query`来执行SQL，得到结果并展示，同时将结果传到`generate_answer`来生成一段总结。

用户在看到结果后可以输入自己的需求生成图片，这里展示了按照Protocol绘制packet数量的饼图`Plot the packet count as a pie chart based on protocol.`得到结果



下图展示了如果用户觉得sql语句不对，输入no则会重新生成sql语句



```
(zrj) ➜  sqlagent  cd /home/hewenyang/zhourunjie/sqlagent ; /usr/bin/env /root/miniconda3/envs/zrj/bin/python /root/.vscode-server/extensions/ms-python.debugpy-20
24.12.0-linux-x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 44749 -- /home/hewenyang/zhourunjie/sqlagent/llmAgent2.py
<IPython.core.display.Image object>
>`How many packets does each flow have?`
switch to database: netflows
{'switch_database': {'database': 'netflows'}}
===========================SQL Query===========================
SELECT src_ip, dst_ip, COUNT(*) as packet_count
FROM packets
GROUP BY src_ip, dst_ip
LIMIT 10;
=================================================================
{'write_query': {'query': 'SELECT src_ip, dst_ip, COUNT(*) as packet_count\nFROM packets\nGROUP BY src_ip, dst_ip\nLIMIT 10;'}}
{'__interrupt__': ()}
Do you want to go to execute query? (yes/no/exit): no
===========================SQL Query===========================
SELECT src_ip, dst_ip, COUNT(*) as packet_count
FROM packets
GROUP BY src_ip, dst_ip
LIMIT 10;
=================================================================
{'write_query': {'query': 'SELECT src_ip, dst_ip, COUNT(*) as packet_count\nFROM packets\nGROUP BY src_ip, dst_ip\nLIMIT 10;'}}
{'__interrupt__': ()}
Do you want to go to execute query? (yes/no/exit): yes
{'execute_query': {'result': "[('192.168.1.1', '192.168.1.2', 1), ('192.168.1.3', '192.168.1.4', 1), ('192.168.1.5', '192.168.1.6', 1), ('192.168.1.7', '192.168.1
.8', 1), ('192.168.1.9', '192.168.1.10', 1), ('192.168.1.11', '192.168.1.12', 1), ('192.168.1.13', '192.168.1.14', 1), ('192.168.1.15', '192.168.1.16', 1), ('192.
168.1.17', '192.168.1.18', 1), ('192.168.1.19', '192.168.1.20', 1)]"}}
Given the following user question, corresponding SQL query, and SQL result, answer the user question.

Question: `How many packets does each flow have?`
SQL Query: SELECT src_ip, dst_ip, COUNT(*) as packet_count
FROM packets
GROUP BY src_ip, dst_ip
LIMIT 10;
SQL Result: [('192.168.1.1', '192.168.1.2', 1), ('192.168.1.3', '192.168.1.4', 1), ('192.168.1.5', '192.168.1.6', 1), ('192.168.1.7', '192.168.1.8', 1), ('192.168
.1.9', '192.168.1.10', 1), ('192.168.1.11', '192.168.1.12', 1), ('192.168.1.13', '192.168.1.14', 1), ('192.168.1.15', '192.168.1.16', 1), ('192.168.1.17', '192.16
8.1.18', 1), ('192.168.1.19', '192.168.1.20', 1)]
{'generate_answer': {'answer': 'Each flow has 1 packet. The result shows pairs of source and destination IPs (representing a flow) along with a count of 1 for eac
h pair, which indicates that in this sample (limited to 10 results), each flow has 1 packet.'}}
{'__interrupt__': ()}
What do you want to plot? (e.g. histogram, scatter plot): `Plot the packet count as a pie chart based on protocol.`
import matplotlib.pyplot as plt


def plot(data):
    labels = ['Protocol']
    sizes = [len(data)]
    plt.pie(sizes, labels=labels, autopct='%1.1f%%')
    plt.axis('equal')
    plt.savefig('plot.png')
    return "plot.png"
{'plot_result': {'imgpth': 'plot.png'}}
(zrj) ➜  sqlagent 
```

llmAgent_PandasAI.py

创建mysql的employees数据库

```
wget 'https://codeload.github.com/datacharmer/test_db/zip/master' -O test_db-master.zip
unzip test_db-master.zip
cd test_db-master
mysql -u root -p88888888 < employees.sql
```

发现pandasai只能接收一张表作为输入，因此如果只用pandasai无法实现多表查询的功能，需要自己搭建agent生成多表查询的sql语句后，才能将多表查询的结果表给pandasai作为输入，得到分析结果和图片

多表查询

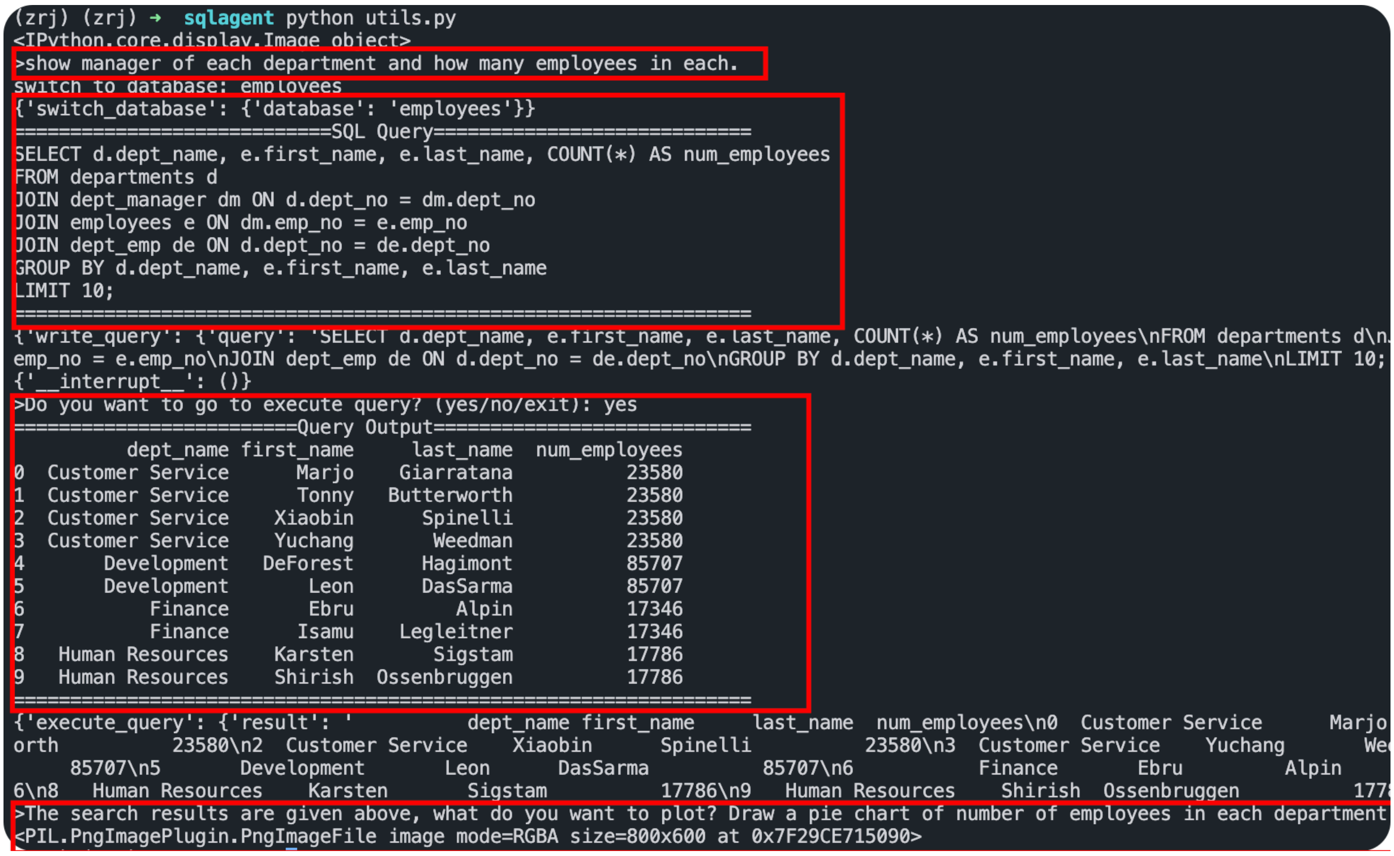
Question:

```
show manager of each department and how many employees in each.
```

LLM监测到该问题对应的数据库应该是`employees`，然后生成的SQL查询语句

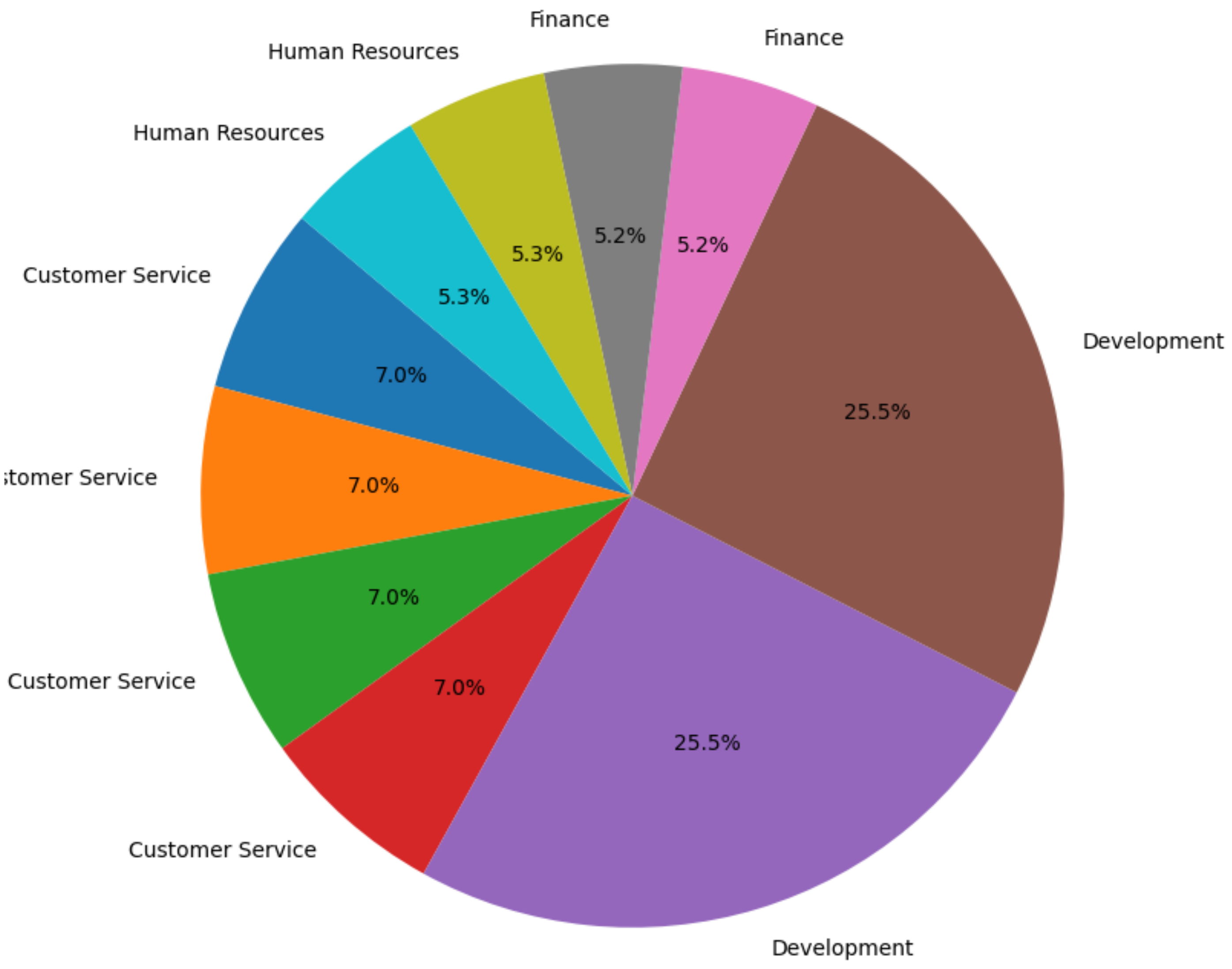
```
SELECT
    d.dept_name,
    e.first_name,
    e.last_name,
    COUNT(de.emp_no) AS num_employees
FROM
```

```
    departments d
        JOIN
    dept_manager dm ON d.dept_no = dm.dept_no
        JOIN
    employees e ON dm.emp_no = e.emp_no
        JOIN
    dept_emp de ON d.dept_no = de.dept_no
GROUP BY d.dept_name, e.first_name, e.last_name
LIMIT 10;
```



```
(zrj) (zrj) ➜  sqlagent python utils.py
<IPython.core.display.Image object>
>show manager of each department and how many employees in each.
switch to database: employees
{'switch_database': {'database': 'employees'}}
============================SQL Query============================
SELECT d.dept_name, e.first_name, e.last_name, COUNT(*) AS num_employees
FROM departments d
JOIN dept_manager dm ON d.dept_no = dm.dept_no
JOIN employees e ON dm.emp_no = e.emp_no
JOIN dept_emp de ON d.dept_no = de.dept_no
GROUP BY d.dept_name, e.first_name, e.last_name
LIMIT 10;
=================================================================
{'write_query': {'query': 'SELECT d.dept_name, e.first_name, e.last_name, COUNT(*) AS num_employees\nFROM departments d\n
emp_no = e.emp_no\nJOIN dept_emp de ON d.dept_no = de.dept_no\nGROUP BY d.dept_name, e.first_name, e.last_name\nLIMIT 10;
{'__interrupt__': ()}
>Do you want to go to execute query? (yes/no/exit): yes
=========================Query Output============================
          dept_name first_name     last_name  num_employees
0  Customer Service      Marjo    Giarratana          23580
1  Customer Service      Tonny   Butterworth          23580
2  Customer Service    Xiaobin      Spinelli          23580
3  Customer Service    Yuchang       Weedman          23580
4       Development   DeForest      Hagimont          85707
5       Development       Leon      DasSarma          85707
6           Finance       Ebru         Alpin          17346
7           Finance      Isamu    Legleitner          17346
8   Human Resources    Karsten       Sigstam          17786
9   Human Resources    Shirish  Ossenbruggen          17786
=================================================================
{'execute_query': {'result': '          dept_name first_name     last_name  num_employees\n0  Customer Service      Marjo
orth          23580\n2  Customer Service    Xiaobin      Spinelli          23580\n3  Customer Service    Yuchang       We
    85707\n5       Development       Leon      DasSarma          85707\n6           Finance       Ebru         Alpin
6\n8   Human Resources    Karsten       Sigstam          17786\n9   Human Resources    Shirish  Ossenbruggen          177
>The search results are given above, what do you want to plot? Draw a pie chart of number of employees in each department
<PIL.PngImagePlugin.PngImageFile image mode=RGBA size=800x600 at 0x7F29CE715090>
```

给出绘图需求`Draw a pie chart of number of employees in each department.`最后绘图结果会被pandasai自动保存在`exports/charts`中



Reference

Build a Question/Answering system over SQL data | 🦜🔗 LangChain

PandasAI - Conversational Data Analysis (pandas-ai.com)