《数据库系统及其应用实践》课程实验报告

学号：10224507002 姓名：唐镱航 完成日期：2025/6/27

实验目标

近年来，大语言模型（如通义千问、ChatGPT 等）已能基于自然语言生成结构化查询语言（SQL），在此背景下，本实验旨在引导学生综合运用数据库知识、服务端开发技能与大模型调用技术，完成一个基于 MCP 协议的数据查询系统，并探索其功能扩展与优化方式。

实验要求

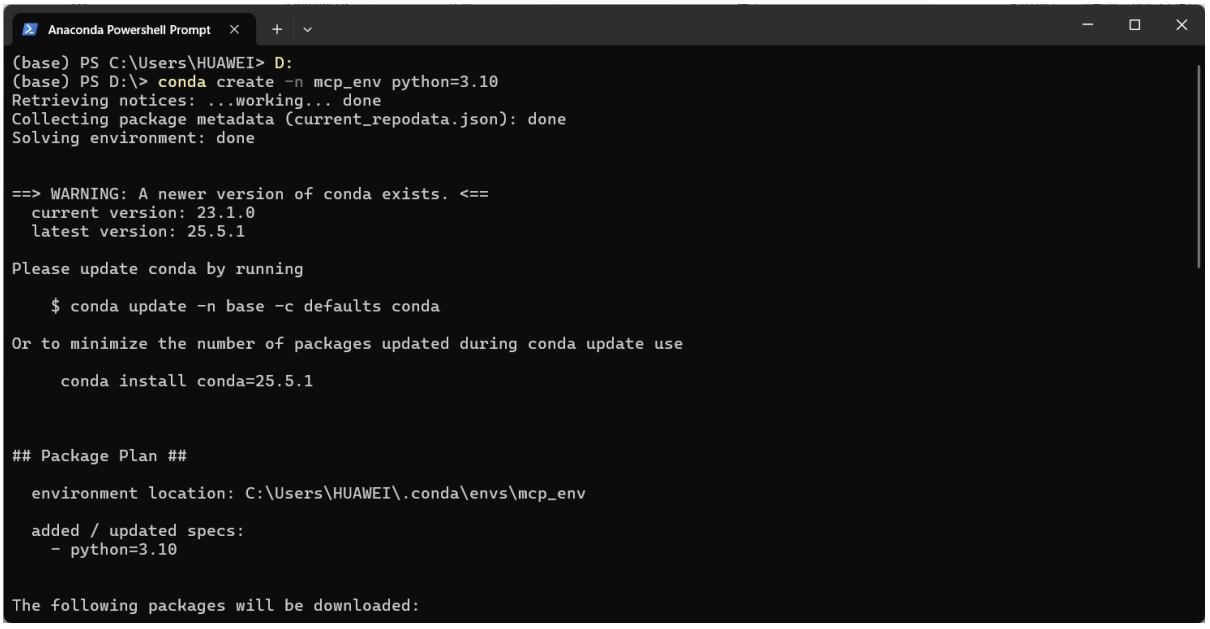
1、按照实验内容，依次完成每个实验步骤；
2、操作实验步骤时，需要理解该操作步骤的目的，预判操作的结果；当操作结果与预判不符时，及时向任课教师和助教咨询；
3、在实验报告中需要包括安装过程截图、测试截图和小项目运行测试截图；
4、对实验中遇到的问题、解决方案及收获进行总结；
5、确保实验报告整洁、美观（注意字体、字号、对齐、截图大小和分页等；）

实验过程记录

1. MCP本地部署

(1) 配置虚拟环境

在Anaconda中运行命令 `conda create -n mcp_env python=3.10` 创建虚拟环境



(2) 激活虚拟环境

运行 `conda activate mcp_env` 激活虚拟环境

```
(base) PS D:\> conda activate mcp_env
(mcp_env) PS D:\>
```

(3) 安装uv

运行 `pip install uv` 安装 uv , 然后可以运行命令 `uv --version` 查看 uv 版本并检查 uv 是否安装成功, 这里运行后可以观察到 uv 版本为 `uv 0.7.14` 因此可以确定 uv 安装成功

```
(base) PS D:\> conda activate mcp_env
(mcp_env) PS D:\>  pip install uv
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting uv
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/af/cc/d16ad99fd666da29cbed13602ac6c6d6e61a5d6307943eea0f400991049a/uv-0.7.14-py3-none-win_amd64.whl (19.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.1/19.1 MB 10.1 MB/s eta 0:00:00
Installing collected packages: uv
Successfully installed uv-0.7.14
(mcp_env) PS D:\> uv --version
uv 0.7.14 (e7f596711 2025-06-23)
```

(4) 克隆MCP项目

使用 `git clone https://github.com/alexcc4/mcp-mysql-server.git` 克隆项目代码

```
(mcp_env) PS D:\> git clone https://github.com/alexcc4/mcp-mysql-server.git
Cloning into 'mcp-mysql-server'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10 (delta 2), reused 10 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (10/10), 12.47 KiB | 4.16 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

(5) 使用uv安装依赖

先切换至项目目录中

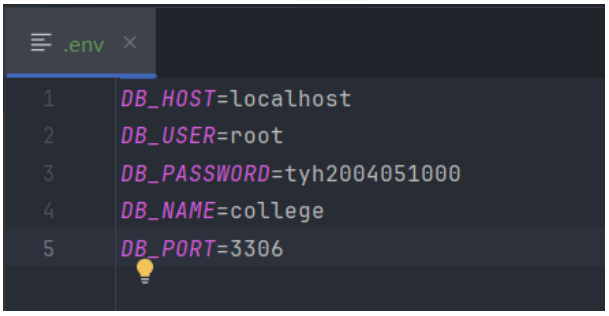
```
(mcp_env) PS D:\> cd mcp*
(mcp_env) PS D:\mcp-mysql-server>
```

然后运行指令 `uv sync` 安装依赖

```
Creating virtual environment at: .venv
Resolved 28 packages in 5ms
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ [0/27] Installing wheels...
warning: Failed to hardlink files; falling back to full copy. This may lead to degraded performance.
         If the cache and target directories are on different filesystems, hardlinking may not be supported.
         If this is intentional, set `export UV_LINK_MODE=copy` or use `--link-mode=copy` to suppress this warning.
Installed 27 packages in 531ms
 + annotated-types==0.7.0
 + anyio==4.8.0
 + certifi==2025.1.31
 + click==8.1.8
 + colorama==0.4.6
 + h11==0.14.0
 + httpcore==1.0.7
 + httpx==0.28.1
 + httpx-sse==0.4.0
 + idna==3.10
 + markdown-it-py==3.0.0
 + mcp==1.3.0
 + mdurl==0.1.2
 + mysqlclient==2.2.7
 + pydantic==2.10.6
 + pydantic-core==2.27.2
 + pydantic-settings==2.8.1
 + pygments==2.19.1
 + python-dotenv==1.0.1
```

(6) 配置环境变量

在项目文件夹中新建 `.env` 文件, 然后设置数据库相关环境变量



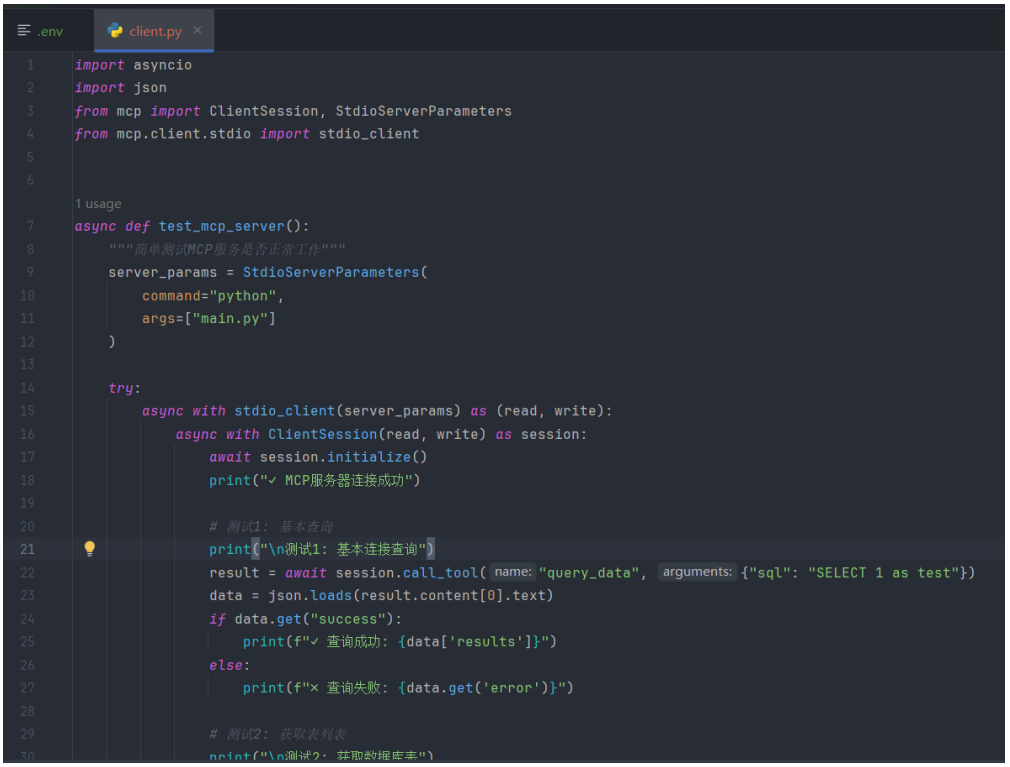
```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=tyh2004051000
DB_NAME=college
DB_PORT=3306
```

修改 `main.py` 中读取环境变量的代码

```python
import dotenv

dotenv.load_dotenv()

# Create MCP server instance
mcp = FastMCP("mysql-server")

# Database connection configuration
DB_CONFIG = {
    "host": os.getenv("DB_HOST", "localhost"),
    "user": os.getenv("DB_USER"),
    "passwd": os.getenv("DB_PASSWORD"),
    "db": os.getenv("DB_NAME"),
    "port": int(os.getenv("DB_PORT", 3306))
}
```

(7) 测试运行

新建一个 `client.py` 文件, 简单编写代码用于测试 `mcp` 是否成功运行



```python
import asyncio
import json
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client


1 usage
async def test_mcp_server():
    """简单测试MCP服务是否正常工作"""
    server_params = StdioServerParameters(
        command="python",
        args=["main.py"]
    )

    try:
        async with stdio_client(server_params) as (read, write):
            async with ClientSession(read, write) as session:
                await session.initialize()
                print("✓ MCP服务器连接成功")

                # 测试1: 基本查询
                print("\n测试1: 基本连接查询")
                result = await session.call_tool( name: "query_data", arguments: {"sql": "SELECT 1 as test"})
                data = json.loads(result.content[0].text)
                if data.get("success"):
                    print(f"✓ 查询成功: {data['results']}")
                else:
                    print(f"× 查询失败: {data.get('error')}")

                # 测试2: 获取表列表
                print("\n测试2: 获取数据库表")
```

运行命令`uv run main.py`以运行`mcp`

```
(base) PS C:\Users\HUAWEI> conda activate mcp_env
(mcp_env) PS C:\Users\HUAWEI> D:
(mcp_env) PS D:\> cd mcp*
(mcp_env) PS D:\mcp-mysql-server> uv run main.py
MySQL MCP server started, connected to localhost:3306/college
MCP server is running...
```

然后运行`client.py`代码测试运行是否成功

```
(mcp_env) PS D:\mcp-mysql-server> python client.py
✓ MCP服务器连接成功

测试1: 基本连接查询
✓ 查询成功: [{'test': 1}]

测试2: 获取数据库表
✓ 数据库: college
✓ 表数量: 13

测试3: 安全检查（应该被拒绝）
✓ 不安全查询被正确拒绝

🎉 所有测试通过，MCP服务正常工作！
(mcp_env) PS D:\mcp-mysql-server>
```

2. 通义 API 调用模块

2.1 模块设计

模块核心功能是通过自然语言输入和数据库结构描述，生成对应的MySQL SQL语句。主要包含以下组件：

1. **API配置**：包括API密钥和端点URL
   - 其中API key为期中项目中申请的密匙，端点URL为通义提供的API文档 https://help.aliyun.com/zh/model-studio/use-qwen-by-calling-api 中提供的使用HTTP方式调用时需配置的endpoint
2. **系统提示词**：定义模型的角色和任务
   - 使用提示词： "你是一个SQL专家。根据用户输入和数据库结构，生成对应的MySQL SQL语句，只返回SQL语句本身。"，用于定义模型的角色，并规定其返回的结果应只有SQL语句本身
3. **核心转换函数**：处理输入并调用API
   - 使用`requests.post`获取返回状态，若状态码为200，则获取json结果，然后获取其中content结果

2.2 关键代码实现

```python
import requests
import json

API_KEY = "sk-bdc3b0de61364bad8bd6944e2394907b"
```

```
API_URL = " https://dashscope.aliyuncs.com/compatible-
mode/v1/chat/completions"

SYSTEM_PROMPT = "你是一个SQL专家。根据用户输入和数据库结构，生成对应的MySQL SQL语
句，只返回SQL语句本身。"

def nl2sql(nl_query: str, schema: dict) -> str:
    schema_str = json.dumps(schema, ensure_ascii=False)
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }
    data = {
        "model": "qwen-turbo",
        "messages": [
            {"role": "system", "content": SYSTEM_PROMPT + f"\n数据库结构:
{schema_str}"},
            {"role": "user", "content": nl_query}
        ]
    }
    response = requests.post(API_URL.strip(), headers=headers, json=data)
    if response.status_code == 200:
        result = response.json()
        return result.get("choices", [{}])[0].get("message",
{}).get("content", "")
    else:
        raise Exception(f"通义API请求失败: {response.status_code},
{response.text}")
```

2.3 功能说明

1. **API调用配置**:
   - 使用 `requests` 库发送HTTP POST请求
   - 包含认证头部(`Authorization`)和内容类型声明
2. **提示工程**:
   - 系统提示明确模型角色为SQL专家
   - 要求只返回SQL语句本身，避免多余解释
   - 将数据库结构以JSON格式传递给模型
3. **错误处理**:
   - 检查HTTP状态码
   - 异常情况下抛出包含详细信息的异常

2.4 大模型测试

由于期中项目已进行对通义千问的测试，所以在此省略测试步骤

3. 查询控制模块

3.1 模块目标

本模块旨在通过提供后端 HTTP 接口，实现对数据库的基础访问控制，包含：

- **数据库结构（Schema）提取**：获取当前数据库中所有表及其字段定义；
- **SQL 查询执行服务**：执行自然语言转换后的 SQL 查询，返回结构化 JSON 结果；
- 提供 **标准化 RESTful API**，供前端 CLI 工具或其他服务调用。

3.2 接口功能描述

1. `/schema [GET]`

- **功能**：提取并返回数据库中所有表的结构定义（表名与字段列表）。
- **输入**：无
- **输出格式**：

```
{
  "success": true,
  "schema": {
    "users": [
      {"Field": "id", "Type": "int", ...},
      {"Field": "name", "Type": "varchar(255)", ...}
    ]
  }
}
```

- **异常情况**：数据库连接失败、权限问题等会返回：

```
{
  "success": false,
  "error": "错误信息"
}
```

2. `/query [POST]`

- **功能**：接收 SQL 查询语句并执行，返回查询结果（支持 SELECT）。
- **输入格式**：

```
{
  "sql": "SELECT * FROM users;"
}
```

- **输出格式**：

```
{
  "success": true,
  "results": [
    {"id": 1, "name": "Alice"},
    {"id": 2, "name": "Bob"}
  ],
  "rowCount": 2
}
```

- **异常情况**：SQL 语法错误、表不存在等将返回：

```
{
  "success": false,
  "error": "错误信息"
}
```

3.3 核心实现逻辑分析

1. Flask 应用初始化与数据库连接

```
from flask import Flask
import MySQLdb

app = Flask(__name__)
```

使用 Flask 构建轻量级 HTTP 服务，并封装了数据库连接函数 `get_connection()`，从 `.env` 文件加载数据库连接参数：

```
def get_connection():
    return MySQLdb.connect(
        host=os.getenv("DB_HOST", "localhost"),
        user=os.getenv("DB_USER"),
        passwd=os.getenv("DB_PASSWORD"),
        db=os.getenv("DB_NAME"),
        port=int(os.getenv("DB_PORT", 3306)),
        charset="utf8mb4"
    )
```

2. Schema 提取接口 `/schema`

```
cursor.execute("SHOW TABLES")
...
cursor.execute(f"DESCRIBE `{table}`")
```

遍历所有表，提取字段信息，返回结构化的 schema 字典。

3. SQL 执行接口 `/query`

```
data = request.get_json()
cursor.execute(sql)
results = cursor.fetchall()
```

执行 SQL 查询并返回 JSON 格式结果，支持 `DictCursor` 让结果以字段名键值形式返回，更适合接口输出。

4. CLI 界面

4.1 模块目标

本模块通过命令行界面（CLI）实现自然语言查询功能，用户可输入自然语言问题，系统自动转换为 SQL 查询语句并返回查询结果。相比基本版本，本模块进一步优化用户交互体验，实现了：

- **颜色高亮输出**
- **表格美观展示查询结果**
- **交互提示语清晰友好**

4.2 功能描述

主要功能包括：

1. **终端交互输入**：用户可连续输入自然语言查询，系统自动应答；
2. **SQL 自动生成**：结合数据库结构使用 `nl2sql` 函数生成查询语句；
3. **语句执行与错误捕获**：执行 SQL 并返回查询结果或失败信息；
4. **界面美化增强体验**：
   - 使用 `colorama` 提供彩色输出；
   - 使用 `tabulate` 美观打印表格数据；
   - 提示语风格统一，增强用户引导。

4.3 关键代码解析

1) 美化库初始化

```
from tabulate import tabulate
from colorama import init, Fore, Style
init(autoreset=True)
```

- `colorama`：控制终端文字颜色；
- `tabulate`：表格化显示查询结果；
- `init(autoreset=True)`：自动重置样式，避免污染后续输出。

2) 欢迎语与退出提示

```
print(Fore.CYAN + "=" * 60)
print(Fore.GREEN + "欢迎使用自然语言转 SQL 查询工具！（输入 exit 或 quit 退出）")
print(Fore.CYAN + "=" * 60)
```

- 增加边框线和颜色高亮，增强第一印象与交互感；
- 提示用户如何退出，提高可用性。

3) 查询输入与转换执行

```
nl_query = input(Fore.YELLOW + "\n请输入您的查询问题： " + Style.RESET_ALL)
sql = nl2sql(nl_query, schema)
print(Fore.BLUE + "\n生成的 SQL 语句：" + Fore.RESET)
print(Fore.LIGHTWHITE_EX + f"{sql}")
```

- 提示颜色使用黄色；
- SQL 输出使用蓝+白，便于用户验证语句正确性。

4. 查询结果展示

```
if result["results"]:
    headers = result["results"][0].keys()
    rows = [row.values() for row in result["results"]]
    print(tabulate(rows, headers=headers, tablefmt="grid"))
else:
    print(Fore.YELLOW + "查询成功，但结果为空。")
```

- 使用 `tabulate` 美化表格；
- 自动提取列名并渲染为网格样式，提升结果可读性；
- 针对空结果给出特殊提示。

5. 错误捕获与反馈

```
except Exception as e:
    print(Fore.RED + f"\n发生错误：{e}")
```

- 捕获运行时异常；
- 使用红色高亮提示错误，增强警示性。

4.4 CLI结果展示

```
============================================================
欢迎使用自然语言转 SQL 查询工具！（输入 exit 或 quit 退出）
============================================================

请输入您的查询问题: What are the names of students who have more than one advisor?

生成的 SQL 语句：
SELECT name
FROM student
WHERE ID IN (
    SELECT s_ID
    FROM advisor
    GROUP BY s_ID
    HAVING COUNT(i_ID) > 1
);

查询成功，结果如下：
查询成功，但结果为空。
```

模块亮点：

- 实现了更人性化的 CLI 界面；
- 使用颜色与表格美化用户交互；
- 实现了容错处理、语义清晰的提示语；
- 架构清晰，易扩展为 GUI 或 Web 端接口。

5. 基础任务测试与结果展示

Question 1: List the names of all courses ordered by their titles and credits. (easy)

```
============================================================
欢迎使用自然语言转 SQL 查询工具！（输入 exit 或 quit 退出）
============================================================

请输入您的查询问题: List the names of all courses ordered by their titles and credits.

生成的 SQL 语句：
SELECT title, credits FROM course ORDER BY title, credits;

查询成功，结果如下：
+--------------------------------------------------+-----------+
| title                                            |   credits |
+==================================================+===========+
| Accounting                                       |         3 |
+--------------------------------------------------+-----------+
| Aerodynamics                                     |         3 |
+--------------------------------------------------+-----------+
| African History                                  |         3 |
+--------------------------------------------------+-----------+
| Animal Behavior                                  |         3 |
+--------------------------------------------------+-----------+
| Antidisestablishmentarianism in Modern America   |         4 |
+--------------------------------------------------+-----------+
| Aquatic Chemistry                                |         3 |
+--------------------------------------------------+-----------+
| Aquatic Chemistry                                |         4 |
+--------------------------------------------------+-----------+
| Arabic                                           |         3 |
```

给出的SQL语句符合问题要求，结果正确

Question 2: What is the title, credit value, and department name for courses with more than one prerequisite? (medium)

```
请输入您的查询问题: What is the title, credit value, and department name for courses with more than one prerequisite?

生成的 SQL 语句:
SELECT c.title, c.credits, c.dept_name
FROM course c
WHERE (
    SELECT COUNT(*)
    FROM prereq p
    WHERE p.course_id = c.course_id
) > 1;

查询成功，结果如下:
+------------------------+-----------+-------------+
| title                  |   credits | dept_name   |
+========================+===========+=============+
| Rock and Roll          |         3 | Marketing   |
+------------------------+-----------+-------------+
| Ponzi Schemes          |         3 | Civil Eng.  |
+------------------------+-----------+-------------+
| Graph Theory           |         3 | Psychology  |
+------------------------+-----------+-------------+
| Operating Systems      |         3 | Psychology  |
+------------------------+-----------+-------------+
| Game Programming       |         4 | Comp. Sci.  |
+------------------------+-----------+-------------+
| Embedded Systems       |         4 | Finance     |
+------------------------+-----------+-------------+
| Immunology             |         3 | Biology     |
```

给出的SQL语句符合问题要求，结果正确

Question 3: What are the names of students who have more than one advisor? (medium)

```
请输入您的查询问题:  What are the names of students who have more than one advisor?

生成的 SQL 语句:
SELECT name
FROM student
WHERE ID IN (
    SELECT s_ID
    FROM advisor
    GROUP BY s_ID
    HAVING COUNT(i_ID) > 1
);

查询成功，结果如下:
查询成功，但结果为空。
```

给出的SQL语句符合问题要求，结果正确

Question 4: What are the titles of courses without prerequisites? (hard)

```
请输入您的查询问题: What are the titles of courses without prerequisites?

生成的 SQL 语句：
SELECT title
FROM course
WHERE course_id NOT IN (SELECT DISTINCT course_id FROM prereq);

查询成功，结果如下：
+-------------------------------------------------+
| title                                           |
+=================================================+
| Diffusion and Phase Transformation              |
+-------------------------------------------------+
| Image Processing                                |
+-------------------------------------------------+
| Differential Equations                          |
+-------------------------------------------------+
| Thermodynamics                                  |
+-------------------------------------------------+
| Differential Geometry                           |
+-------------------------------------------------+
| Manufacturing                                   |
+-------------------------------------------------+
| Number Theory                                   |
+-------------------------------------------------+
| Romantic Literature                             |
+-------------------------------------------------+
```

给出的SQL语句符合问题要求，结果正确

Question 5: Give the name and building of the departments with greater than average budget. (extra)

```
请输入您的查询问题:  Give the name and building of the departments with greater than average budget.

生成的 SQL 语句：
SELECT dept_name, building
FROM department
WHERE budget > (SELECT AVG(budget) FROM department);

查询成功，结果如下：
+-------------+-------------+
| dept_name   | building    |
+=============+=============+
| Astronomy   | Taylor      |
+-------------+-------------+
| Athletics   | Bronfman    |
+-------------+-------------+
| Biology     | Candlestick |
+-------------+-------------+
| Cybernetics | Mercer      |
+-------------+-------------+
| English     | Palmer      |
+-------------+-------------+
| Finance     | Candlestick |
+-------------+-------------+
| History     | Taylor      |
+-------------+-------------+
| Languages   | Linderman   |
+-------------+-------------+
| Math        | Brodhead    |
```

给出的SQL语句符合问题要求，结果正确

Question 6: What is the name of the department with the most credits? (hard)

```
请输入您的查询问题: What is the name of the department with the most credits?

生成的 SQL 语句:
SELECT dept_name
FROM department
ORDER BY budget DESC
LIMIT 1;

查询成功，结果如下:
+-------------+
| dept_name   |
+=============+
| Physics     |
+-------------+
```

给出的SQL语句没有考虑同时有多个人最多的情况，仅用`LIMIT 1;`取一个人，结果错误

Question 7: Give the title of the prerequisite to the course International Finance. (hard)

```
请输入您的查询问题:  Give the title of the prerequisite to the course International Finance.

生成的 SQL 语句:
SELECT c.title
FROM course c
JOIN prereq p ON c.course_id = p.prereq_id
WHERE p.course_id = (SELECT course_id FROM course WHERE title = 'International Finance');

查询失败: (1242, 'Subquery returns more than 1 row')
```

错误信息`(1242, 'Subquery returns more than 1 row')`表明:

1. 子查询`(SELECT course_id FROM course WHERE title = 'International Finance')`返回了多个 course_id
2. 但`p.course_id = (...)`这种等值比较只能处理单值
   所以应该把`WHRER`后的`=`换成`IN`才对

6. MCP功能增强任务

6.1 查询日志记录 /logs

为增强系统的可追溯性与可维护性，设计并实现了查询日志记录机制。本模块自动记录所有被执行的 SQL 查询语句，并提供`/logs`接口用于查看日志内容，实现以下目标:

- **记录历史查询内容与时间**;
- **便于调试与教学演示**;
- **为未来的模型调优和权限审计提供基础数据支撑**

功能	描述
查询日志记录	每次执行 SQL 查询时将语句及时间戳写入 `logs/query.log`
日志读取接口	通过 GET `/logs` 接口返回日志文件内容（纯文本）

1. **日志记录函数 `log_query(sql)`**

```
def log_query(sql):
    with log_lock:
        with open(LOG_FILE, "a", encoding="utf-8") as f:
            f.write(f"[{datetime.datetime.now().isoformat()}]\n{sql}\n\n")
```

- 使用 `threading.Lock()` 实现线程安全写入；
- 每条日志包括时间戳与 SQL 语句，便于定位；
- 日志写入目录为 `logs/query.log`，首次运行自动创建文件夹。

2. **在 `/query` 中调用记录**

```
sql = data.get("sql")
log_query(sql)
```

- 每次执行 SQL 前先记录日志；
- 日志独立于数据库操作流程，防止因查询失败影响记录。

3. **`/logs` 接口实现**

```
@app.route('/logs', methods=['GET'])
def get_logs():
    try:
        with open(LOG_FILE, "r", encoding="utf-8") as f:
            content = f.read()
        return content, 200, {"Content-Type": "text/plain; charset=utf-8"}
    except FileNotFoundError:
        return "No logs found.", 200, {"Content-Type": "text/plain;
charset=utf-8"}
```

- 可通过浏览器或命令行工具（如 curl）访问日志；
- 如果日志文件不存在，将返回“无日志”。

4. **运行示例**

```
[2025-06-26T15:51:11.862271]
SELECT dept_name, building
FROM department
WHERE budget > (SELECT AVG(budget) FROM department);

[2025-06-26T15:51:26.676866]
SELECT title
FROM course
WHERE course_id NOT IN (SELECT DISTINCT course_id FROM prereq);
```

6.2 查询结果分页

为提升接口的实用性与性能表现，系统新增了**分页支持功能。**在数据量较大的场景下，分页可以有效控制返回结果的体积，减少网络传输负担，同时提升用户交互体验与可读性。

新增的分页功能集成在 `/query` 接口中，前端或调用方可以通过可选参数 `page` 和 `pageSize` 控制返回结果的页码与条数。

1. **参数提取与默认处理**

```python
page = int(data.get("page", 1))
page_size = int(data.get("pageSize", 50))
```

- 若未提供分页参数，默认返回第1页，每页50条；
- 使用 `int()` 类型转换，确保安全。

2. **执行 SQL 并分页切片**

```python
cursor.execute(sql)
results = cursor.fetchall()
start = (page - 1) * page_size
end = start + page_size
paged_results = results[start:end]
```

- 分页在原始结果集上完成；
- 返回分页结果与总记录条数。

3. **接口响应格式扩展**

```python
return jsonify({
    "success": True,
    "results": paged_results,
```

```
    "rowCount": len(results),
    "page": page,
    "pageSize": page_size
})
```

- 除结果集外返回当前页码、分页大小与总行数；
- 保持接口兼容性，便于前端展示页数或加载更多。

4. **运行示例**
   分页操作：

```
+-----------+-------------------------------------------------+
|         3 | Assembly Language Programming                   |
+-----------+-------------------------------------------------+
|         3 | Astronautics                                    |
+-----------+-------------------------------------------------+
输入 next 查看下一页，exit 返回主菜单，quit 退出程序: next

第 2 页 / 共 20 页：
+-----------+----------------------+
|   credits | title                |
+===========+======================+
|         4 | Astronomy            |
+-----------+----------------------+
|         4 | Automobile Mechanics |
+-----------+----------------------+
|         4 | Bacteriology         |
+-----------+----------------------+
```

   退出分页，继续下一个问题：

```
+-----------+-----------------------+
|         3 | C  Programming        |
+-----------+-----------------------+
|         3 | C  Programming        |
+-----------+-----------------------+
输入 next 查看下一页，exit 返回主菜单，quit 退出程序: exit

已返回主菜单。

请输入您的查询问题：
```

6.3 表结构简化输出

在数据库表数量较多或结构复杂的场景中，直接返回所有表结构信息可能导致响应体过大、信息冗余。因此本模块新增对 `/schema` 接口的增强：**支持通过查询参数按表名过滤返回特定表的结构信息**，提升接口的实用性与响应效率。

1. **功能描述**
   - 原 `/schema` 接口默认返回**所有表结构**；
   - 新增支持 `GET /schema?table=table_name` 请求格式；
   - 若请求中提供了 `table` 参数，接口仅返回该表的结构信息；
   - 若表不存在，返回 `404` 状态码及提示信息。

2. **接口设计与实现**

CLI 工具中新增功能选项：

```
请选择功能：
  1. 输入自然语言查询
  2. 显示所有表名及其模式
  3. 按表名查看表结构
  quit. 退出程序
```

由 `show_table_schema_by_name()` 函数实现调用：

`resp = requests.get(f"{MCP_URL}/schema", params={"table": table_name})`

若返回成功，解析响应并使用 `tabulate` 格式化输出。

服务器改造：

```python
@app.route('/schema', methods=['GET'])
def schema():
    table_name = request.args.get("table")
    ...
    if table_name:
        # 查询特定表结构
        cursor.execute(f"DESCRIBE `{table_name}`")
        columns = cursor.fetchall()
        return jsonify({"success": True, "schema": {table_name: columns}})
    else:
        # 查询所有表结构（原逻辑）
        ...
```

3. **实现演示**

主菜单：

```
============================================================
 欢迎使用自然语言转 SQL 查询工具!
============================================================

正在连接 MCP 服务并获取数据库结构...
数据库结构获取成功!
------------------------------------------------------------

请选择功能：
  1. 输入自然语言查询
  2. 显示所有表名及其模式
  3. 按表名查看表结构
  quit. 退出程序

------------------------------------------------------------
请输入选项编号: 1
```

自然语言查询：

```
------------------------------------------------------------

请输入选项编号: 1

请输入您的查询问题(exit 返回主菜单, quit 退出程序): 学生

生成的 SQL 语句:
SELECT * FROM student;

查询成功, 结果如下:

第 1 页 / 共 200 页:
```

退出自然语言查询:

```
输入 next 查看下一页, exit 继续下一次查询, quit 退出程序: exit

已切换下一次查询。

请输入您的查询问题(exit 返回主菜单, quit 退出程序): exit

已返回主菜单。
------------------------------------------------------------------


请选择功能:
  1. 输入自然语言查询
  2. 显示所有表名及其模式
  3. 按表名查看表结构
  quit. 退出程序


------------------------------------------------------------------
```

显示所有表:

```
------------------------------------------------------------
请输入选项编号: 2
正在获取所有表名及其模式...

表名: advisor
+-----------+---------+---------+-------+--------+------------+
| Default   | Extra   | Field   | Key   | Null   | Type       |
+===========+=========+=========+=======+========+============+
|           |         | s_ID    | PRI   | NO     | varchar(5) |
+-----------+---------+---------+-------+--------+------------+
|           |         | i_ID    | MUL   | YES    | varchar(5) |
+-----------+---------+---------+-------+--------+------------+

表名: classroom
+-----------+---------+-------------+-------+--------+-------------+
| Default   | Extra   | Field       | Key   | Null   | Type        |
+===========+=========+=============+=======+========+=============+
|           |         | building    | PRI   | NO     | varchar(15) |
+-----------+---------+-------------+-------+--------+-------------+
```

表结构简化输出：

```
------------------------------------------------------------
请输入选项编号：3
请输入要查询的表名：time_slot

表名：time_slot
+-----------+---------+--------------+-------+--------+------------+
| Default   | Extra   | Field        | Key   | Null   | Type       |
+===========+=========+==============+=======+========+============+
|           |         | time_slot_id | PRI   | NO     | varchar(4) |
+-----------+---------+--------------+-------+--------+------------+
|           |         | day          | PRI   | NO     | varchar(1) |
+-----------+---------+--------------+-------+--------+------------+
|           |         | start_hr     | PRI   | NO     | int        |
+-----------+---------+--------------+-------+--------+------------+
|           |         | start_min    | PRI   | NO     | int        |
+-----------+---------+--------------+-------+--------+------------+
|           |         | end_hr       |       | YES    | int        |
+-----------+---------+--------------+-------+--------+------------+
```

7. MCP安全控制任务

7.1 只读 SQL 白名单过滤

在main.py代码中加入函数：

```
def is_safe_select(sql):
    # 只允许以SELECT开头，忽略前导空白和大小写
    sql_strip = sql.strip().lower()
    return sql_strip.startswith('select')
```

然后在 `/query` 中使用该函数检测以实现白名单操作：

```
if not is_safe_select(sql):
        return jsonify({"success": False, "error": "只允许SELECT语句，禁止非只读操作。"}), 403
```

- MCP /query 接口现在只允许以 SELECT 开头的 SQL 语句（忽略前导空白和大小写）。
- 非 SELECT 语句（如 INSERT、UPDATE、DELETE、DROP 等）会被直接拒绝，并返回错误提示：“只允许SELECT语句，禁止非只读操作。”

示例：

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序)：增加一个学生 学号为1 姓名为ttt

生成的 SQL 语句：
INSERT INTO student (ID, name) VALUES ('1', 'ttt');
查询失败：只允许SELECT语句，禁止非只读操作。
```

7.2 关键字段访问控制

为防止用户通过自然语言查询系统访问涉及隐私或敏感信息的数据库字段，本模块新增**关键字段访问控制机制**。系统将自动检测 SQL 查询中是否涉及指定敏感字段（如 `password`、`salary` 等），一经发现即拒绝执行，并返回错误："禁止查询敏感字段（如 password、salary 等）。"，确保数据安全与合规性。

功能通过两步校验保障安全：

1. **查询类型限制**：只允许执行只读类型的 `SELECT` 查询，阻断 `UPDATE`、`DELETE` 等写操作；
2. **敏感字段过滤**：对 SQL 中的 `SELECT` 字段列表进行提取和匹配，若包含敏感字段则拒绝执行。

判断是否为只读 SELECT 查询

```
def is_safe_select(sql):
    sql_strip = sql.strip().lower()
    return sql_strip.startswith('select')
```

- 忽略大小写与前导空格；
- 非 SELECT 查询一律拒绝执行并返回 `403` 状态码。

提取字段名并匹配敏感字段

```
SENSITIVE_FIELDS = {"password", "salary"}

def contains_sensitive_field(sql):
    m = re.match(r"\s*select\s+(.*?)\s+from\s", sql.lower(), re.DOTALL)
    if not m:
        return False
    fields = [f.strip().strip('`"') for f in m.group(1).split(',')]
    for field in fields:
        for sensitive in SENSITIVE_FIELDS:
            if sensitive in field:
                return True
    return False
```

- 使用正则提取 `SELECT ... FROM ...` 中的字段列表；
- 若任意字段名包含敏感关键词，则返回 True，系统中断查询。
- 如需扩展更多敏感字段，只需在 SENSITIVE_FIELDS 集合中添加即可。例如：`SENSITIVE_FIELDS = {"password", "salary", "ssn", "credit_card"}`

示例：

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序)：输出每个教师的工资

生成的 SQL 语句：
SELECT ID, salary FROM instructor;
查询失败：禁止查询敏感字段（如 password、salary 等）。
```

7.3 简易 SQL 注入防御机制

为了防范自然语言转 SQL 查询系统中可能出现的**SQL 注入攻击**，本模块通过关键模式检测，拦截常见的注入方式（如逻辑拼接、关键词欺骗、联合查询、时间盲注等），在 SQL 执行前对输入进行预检测，从而提升系统的安全防护能力。

系统通过正则表达式规则，对可疑 SQL 特征进行模式识别，若匹配成功，将直接拒绝请求并返回："检测到疑似SQL注入攻击，已拦截。"，返回 403 错误。

已涵盖的注入特征包括：

注入类型	检测内容
逻辑注入	`OR`，`AND`，`1=1`，`0=0` 等条件恒真语句
拼接注入	`' + '`，`" + "`，变量拼接等字符串技巧
注释注入	`--`，`#`，`/* ... */`
联合查询注入	`UNION`，`SELECT ... SELECT`
盲注行为	`SLEEP`，`BENCHMARK` 等延时函数
文件读取	`LOAD_FILE`，`INTO OUTFILE` 等

1. **模式定义（`INJECTION_PATTERNS`）**

```
INJECTION_PATTERNS = [
    r"(--|#|/\*.*?\*/)",  # 注释符号
    r";\s*(select|update|delete|insert|\bdrop\b|\bcreate\b)",  # 多语句执行符
    r"\b(OR|AND)\b\s+[^=]*=",  # OR/AND 语句绕过
    r"\b(UNION|SLEEP|BENCHMARK|LOAD_FILE|INTO OUTFILE)\b",  # 高危操作
    r"\b1\s*=\s*1\b", r"\b0\s*=\s*0\b",  # 恒等逻辑
    r"['\"]\s*\+\s*['\"]",  # 字符串拼接（如 'a' + 'b'）
]
```

2. **SQL 检测逻辑**

```
def is_sql_injection(sql):
    sql_lower = sql.lower()
    for pattern in INJECTION_PATTERNS:
        if re.search(pattern, sql_lower, re.DOTALL):
```

```
            return True
    return False
```

3. **拦截接口集成（`/query` 接口）**

```
if is_sql_injection(sql):
    return jsonify({"success": False, "error": "检测到疑似SQL注入攻击，已拦截。"}), 403
```

示例：

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序)：查询所有用户信息，条件是1=1

生成的 SQL 语句：
SELECT * FROM employee WHERE 1=1;
查询失败：检测到疑似SQL注入攻击，已拦截。
```

8. 大模型优化任务

本实验旨在通过优化 Prompt 模板，提高大模型生成 SQL 语句的准确性、鲁棒性与执行效率，进一步提升自然语言到结构化查询语言（NL2SQL）的转换质量。

8.1 优化目标

1. **提升 SQL 转换的准确率（≥10%）**
2. **支持更复杂的自然语言需求（多轮、多样表达）**
3. **输出更高效、执行计划更优的 SQL 语句**

8.2 优化方法

本实验采用了以下三项关键策略进行 Prompt 优化：

1）明确的系统提示（System Prompt）

通过设置专业化系统角色 `你是一个专业的SQL专家`，限定生成任务为 `MySQL SELECT语句转换`，进一步明确目标、规范生成范围，避免生成 UPDATE/DELETE 等无关语句。

同时嵌入结构化规则说明，包括：

- 字段与表名使用反引号
- 字符串值使用单引号
- 使用 JOIN 而非嵌套子查询优先实现连接逻辑
- 多个注意事项指导避免常见错误（如子查询返回多行）

Prompt 核心代码设计

代码通过 `format(schema_info=schema_str)` 将数据库结构动态注入模板，再结合用户自然语言查询和 `qwen-turbo` 模型交互，完整代码如下片段所示：

```
full_prompt = SYSTEM_PROMPT.format(schema_info=schema_str)
...
"messages": [
    {"role": "system", "content": full_prompt},
    {"role": "user", "content": nl_query}
]
```

2）示例增强 Few-shot Learning

Prompt 中引入 6 组精心设计的 Few-shot 示例，包括：

- 基础查询（如字段选择、单表筛选）
- 多表连接（JOIN）
- 聚合与分组（GROUP BY, HAVING）
- 极值处理（MAX + 子查询或窗口函数）
- 多重嵌套子查询（如课程前置结构）

这些示例覆盖了教学数据库中常见结构与自然语言表达，有助于提升模型泛化能力与准确率。

3）SQL 生成效率提示（执行计划简化建议）

在提示中显式强调：

- 优先使用 JOIN 而非嵌套查询（提升性能）
- 极值处理时避免 `LIMIT 1` 简单裁剪，防止遗漏并列最大值
- 提示避免子查询返回多行的错误，用 IN/EXISTS 替代 =
- 强调多行结果容错设计，提升稳定性与可执行性

这些优化可以显著提升 SQL 的运行效率，减少错误率。

8.3 实验效果

用上面第五部分的两个错误问题作为测试：

`What is the name of the department with the most credits?`

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序):  What is the name of the department with the most credits?

生成的 SQL 语句：
SELECT `dept_name`
FROM `course`
GROUP BY `dept_name`
HAVING SUM(`credits`) = (
    SELECT MAX(total_credits)
    FROM (
        SELECT SUM(`credits`) as total_credits
        FROM `course`
        GROUP BY `dept_name`
    ) as subquery
)

查询成功，结果如下：

第 1 页 / 共 1 页：
+-------------+
| dept_name   |
+=============+
| Cybernetics |
+-------------+

已显示全部结果。
```

可以看到本次查询并没有使用`LIMIT 1`，查询正确

`Give the title of the prerequisite to the course International Finance.`

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序): Give the title of the prerequisite to the course International Finance

生成的 SQL 语句：
SELECT `title`
FROM `course`
WHERE `course_id` IN (
    SELECT `prereq_id`
    FROM `prereq`
    WHERE `course_id` IN (
        SELECT `course_id`
        FROM `course`
        WHERE `title` = 'International Finance'
    )
);

查询成功，结果如下：

第 1 页 / 共 1 页：
+--------------------+
| title              |
+====================+
| Elastic Structures |
+--------------------+
```

可以看到本次查询正确使用了IN，没有导致出现`查询失败：(1242, 'Subquery returns more than 1 row')`的情况了

在若干自然语言查询样本中测试发现，优化后的Prompt能有效生成：

- **语法规范**：符合MySQL格式，字段与表名自动加反引号；

- **逻辑正确**：能识别自然语言中的隐含关系，如“选修某课程的学生”需JOIN三张表；
- **极值处理稳健**：能用子查询返回所有并列最高/最多情况，避免遗漏；
- **异常规避**：避免 `=` 处理多行子查询返回的问题，自动使用 `IN` 或 `EXISTS`。

与未优化Prompt相比，该模板在准确率与可解释性方面均有明显提升。

9. GUI 界面

本部分实验实现了一个基于 **Streamlit 框架**的图形化用户界面（GUI），支持用户通过自然语言输入查询意图，自动生成 SQL 并展示查询结果。该界面简洁直观，极大提升了系统的可用性与交互性。

9.1 界面功能

- 支持用户输入自然语言查询问题；
- 实时展示模型生成的 SQL 查询语句；
- 自动向后端 MCP 发送 SQL 请求，返回并展示结果表格；
- 提供数据库表结构的辅助查询功能，提升用户理解；
- 界面美观、交互友好，支持中文提示与状态反馈。

9.2 技术实现

- 使用 Streamlit 作为前端开发框架，快速构建 Web 应用；
- 后端接口通过 `requests` 调用 MCP 查询服务；
- 使用 `pandas` 进行表格展示和结果格式化；
- 支持三类主功能：
  - 自然语言转 SQL 查询
  - 显示所有表结构
  - 查询指定表结构

首先使用指令 `pip install streamlit` 安装 `streamlit`

```
(mcp_env) PS D:\mcp-mysql-server> pip install streamlit
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting streamlit
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/84/3b/35400175788cdd6a43c90dce1e7f567eb6843a3ba0612508c0f19ee31f5f/streamlit-1.46.1-py3-none-any.whl (10.1 MB)
     ---------------------------------------- 10.1/10.1 MB 9.3 MB/s eta 0:00:00
Collecting altair<6,>=4.0 (from streamlit)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/aa/f3/0b6ced594e51cc95d8c1fc1640d3623770d01e4969d29c0bd09945fafefa/altair-5.5.0-py3-none-any.whl (731 kB)
     ---------------------------------------- 731.2/731.2 kB 10.0 MB/s eta 0:00:00
Requirement already satisfied: blinker<2,>=1.5.0 in c:\users\huawei\appdata\roaming\python\python310\site-packages (from streamlit) (1.9.0)
Collecting cachetools<7,>=4.0 (from streamlit)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/00/f0/2ef431fe4141f5e334759d73e81120492b23b2824336883a91ac04ba710b/cachetools-6.1.0-py3-none-any.whl (11 kB)
Requirement already satisfied: click<9,>=7.0 in c:\users\huawei\appdata\roaming\python\python310\site-packages (from streamlit) (8.1.7)
Collecting numpy<3,>=1.23 (from streamlit)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/a3/dd/4b822569d6b96c39d1215dbae0582fd99954dcbcf0c1a13c61783feaca3f/numpy-2.2.6-cp310-cp310-win_amd64.whl (12.9 MB)
     ---------------------------------------- 12.9/12.9 MB 9.7 MB/s eta 0:00:00
Collecting packaging<26,>=20 (from streamlit)
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/20/12/38679034af332785aac8774540895e234f4d07f7545804097de4b666afd8/packaging-25.0-py3-none-any.whl (66 kB)
Collecting pandas<3,>=1.4.0 (from streamlit)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/26/fa/8eeb2353f6d40974a6a9fd4081ad1700e2386cf4264a8f28542fd10b3e38/pandas-2.3.0-cp310-cp310-win_amd64.whl (11.1 MB)
```

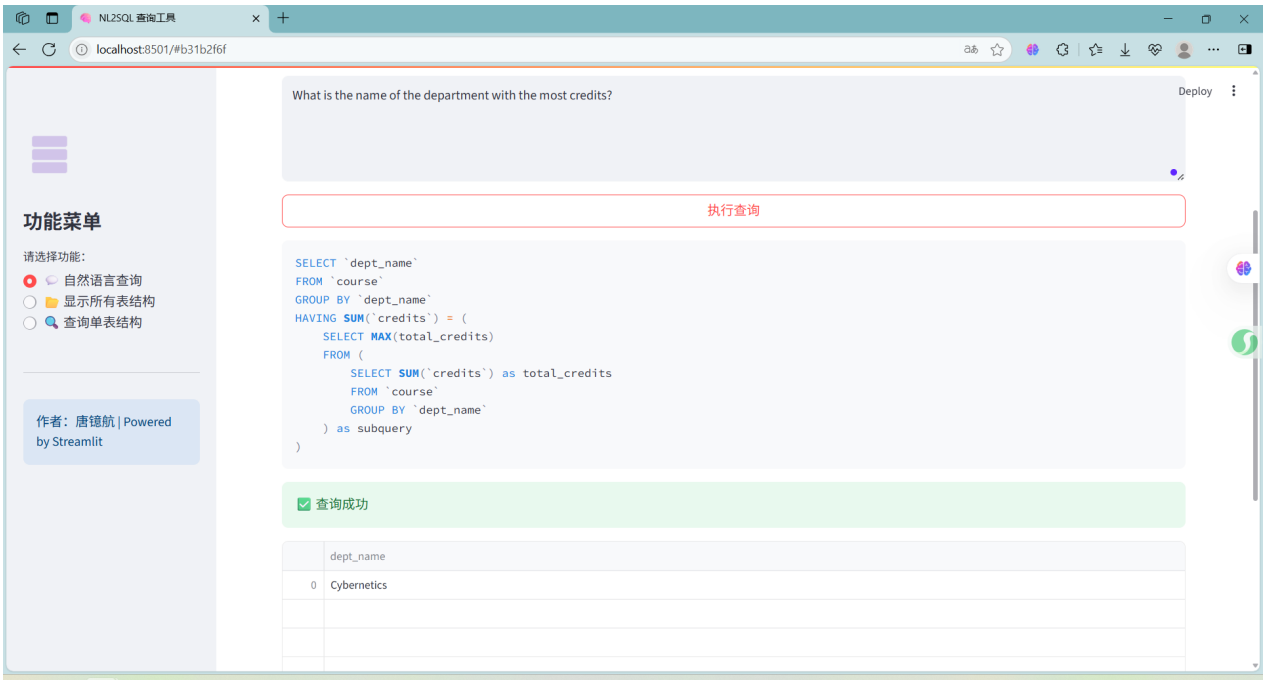
然后创建一个gui_app.py文件用于使用GUI

9.3 核心界面展示

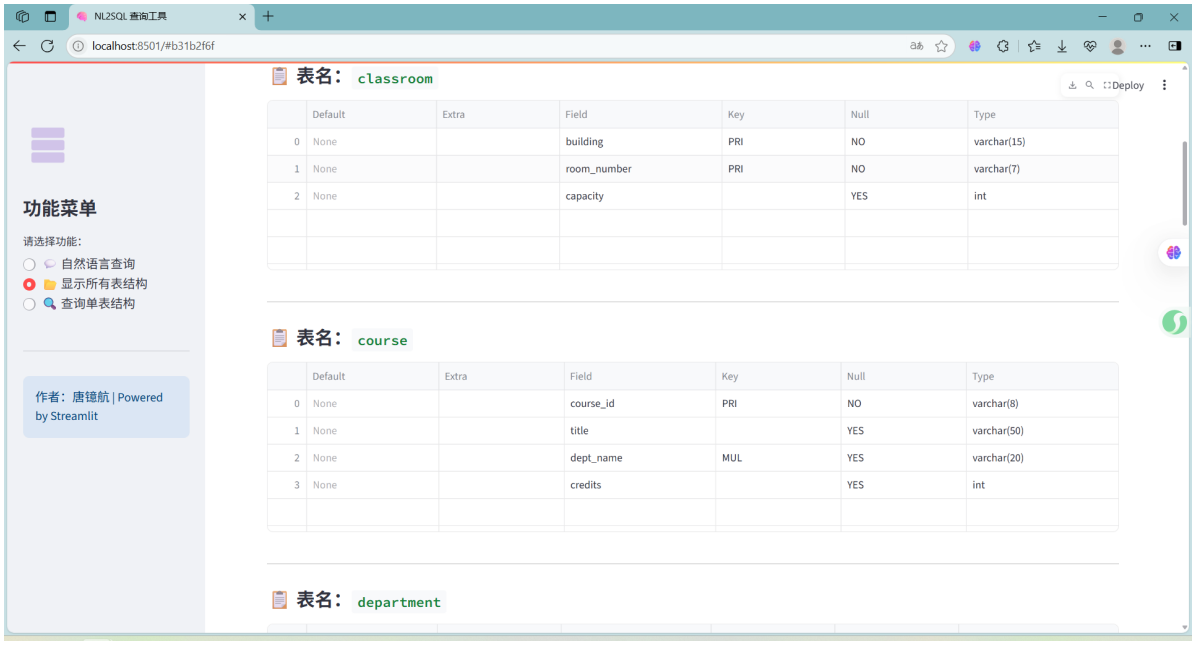
- **顶部横幅（Banner）**：使用渐变色块突出系统主题。

- **侧边栏菜单**：功能选择按钮以图标+文字呈现，界面分组清晰。可自主选择收起或展开

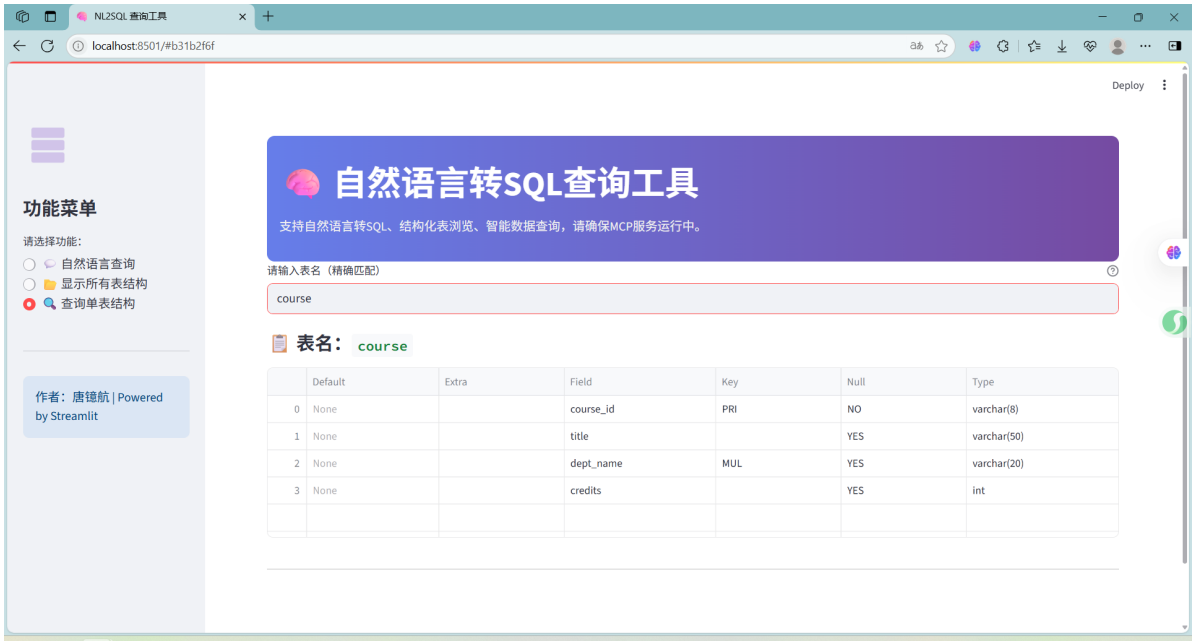
- **自然语言查询模块**：



  - 输入框支持中文提示；
  - 查询按钮触发后展示 SQL 与执行结果；
  - 支持错误信息展示与状态 loading 动画。
- **结构浏览模块**：
  - 可快速查看所有表及字段；

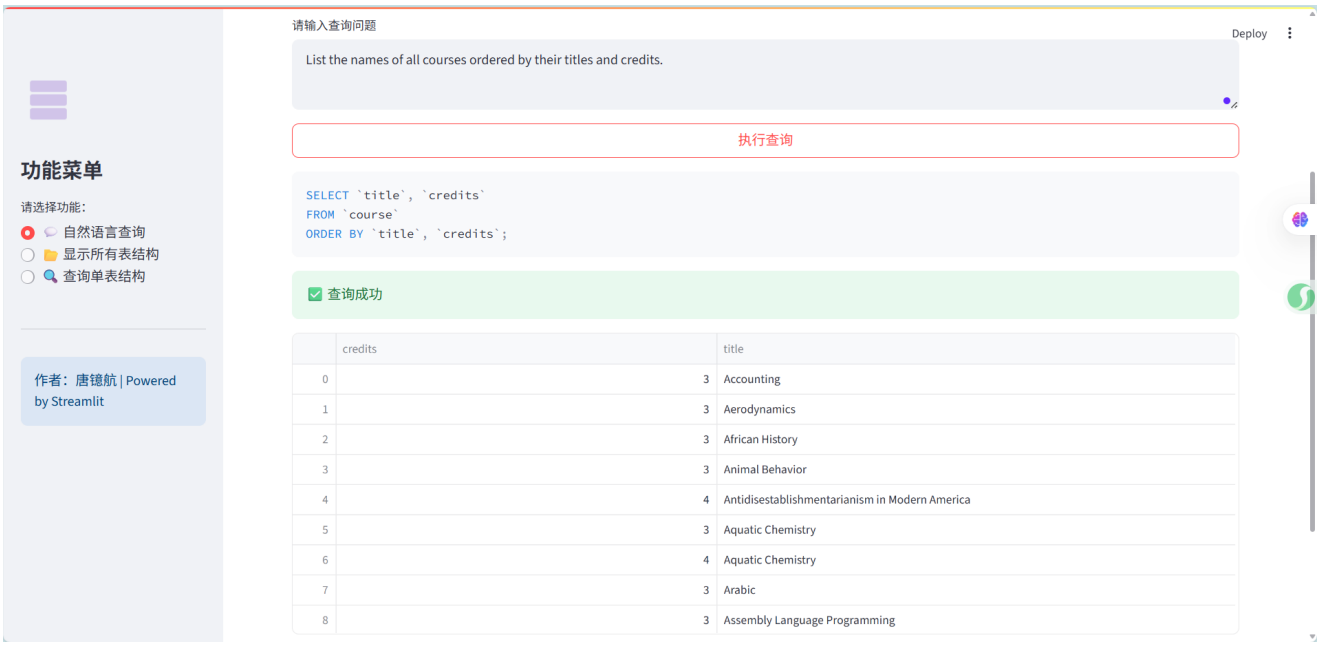


- 支持按表名精确检索，辅助用户理解数据库结构。



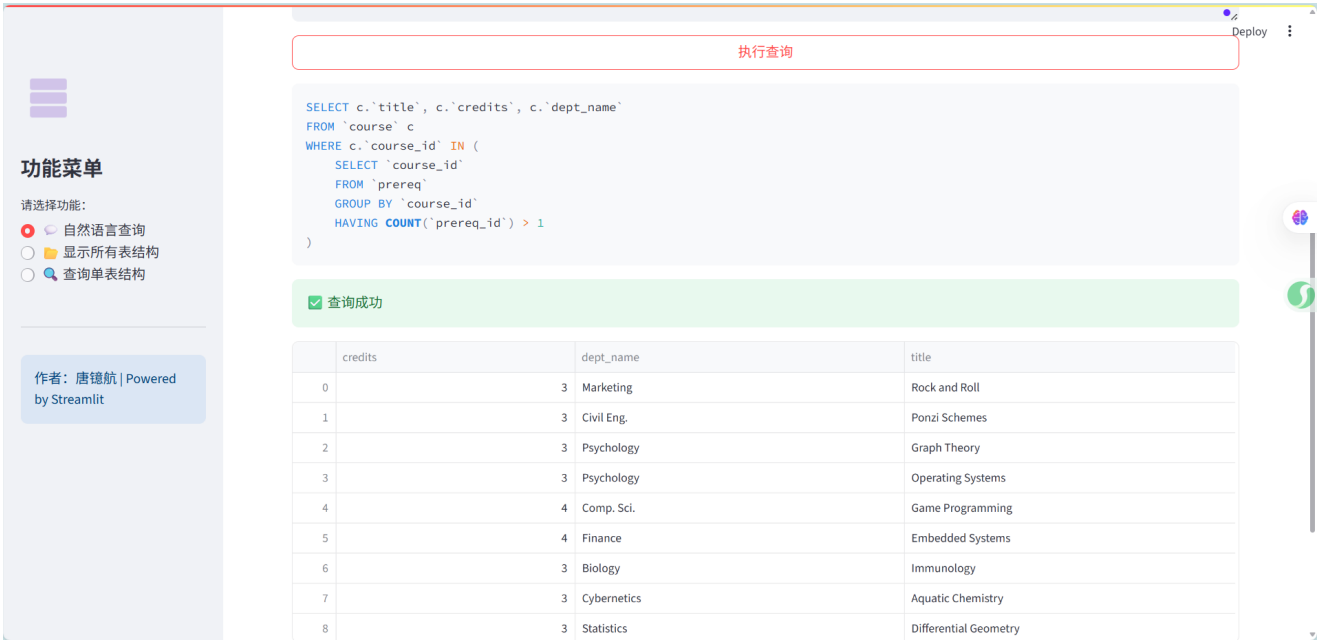
10. 最后测试与结果展示

Question 1: List the names of all courses ordered by their titles and credits.（easy)



给出的SQL语句符合问题要求，结果正确

Question 2: What is the title, credit value, and department name for courses with more than one prerequisite?（medium)

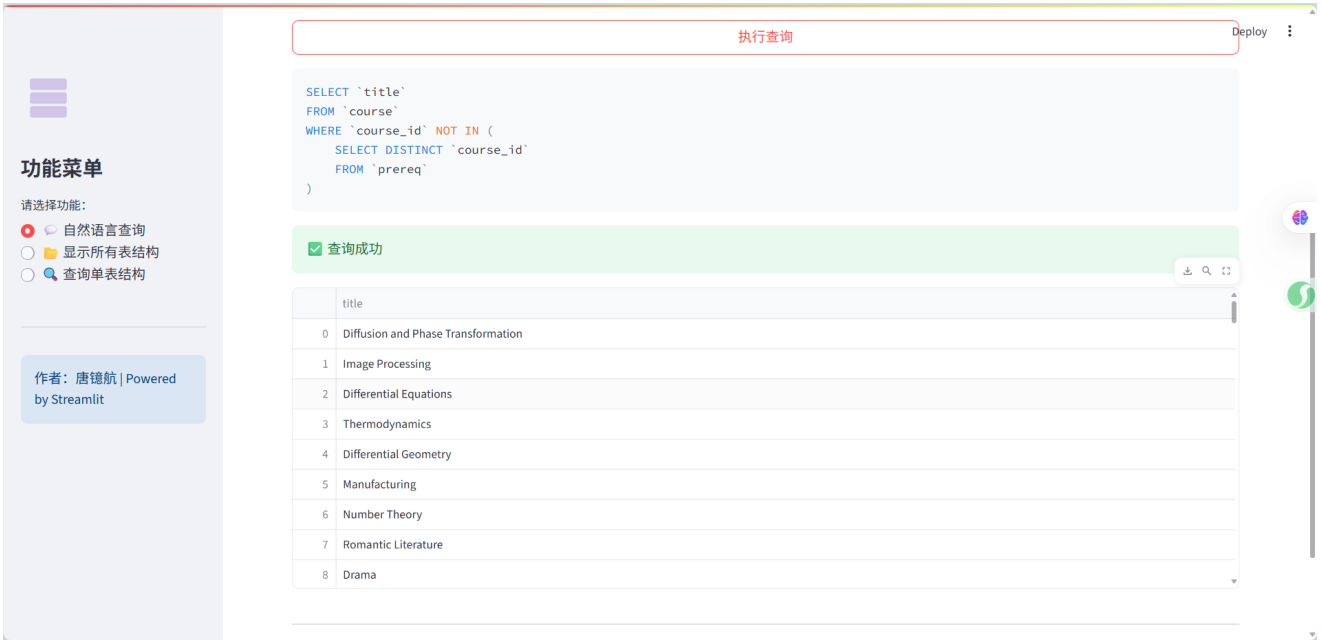


给出的SQL语句符合问题要求，结果正确

Question 3: What are the names of students who have more than one advisor? (medium)

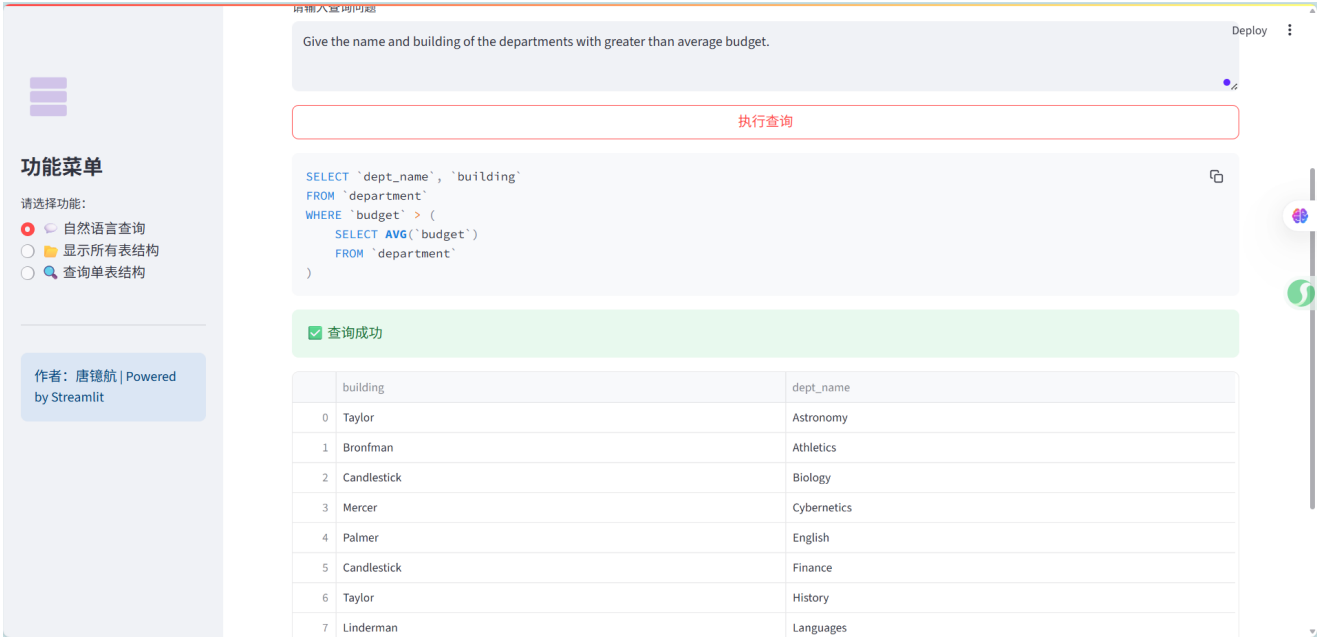
给出的SQL语句符合问题要求，结果正确

Question 4: What are the titles of courses without prerequisites? (hard)



给出的SQL语句符合问题要求，结果正确

Question 5: Give the name and building of the departments with greater than average budget. (extra)

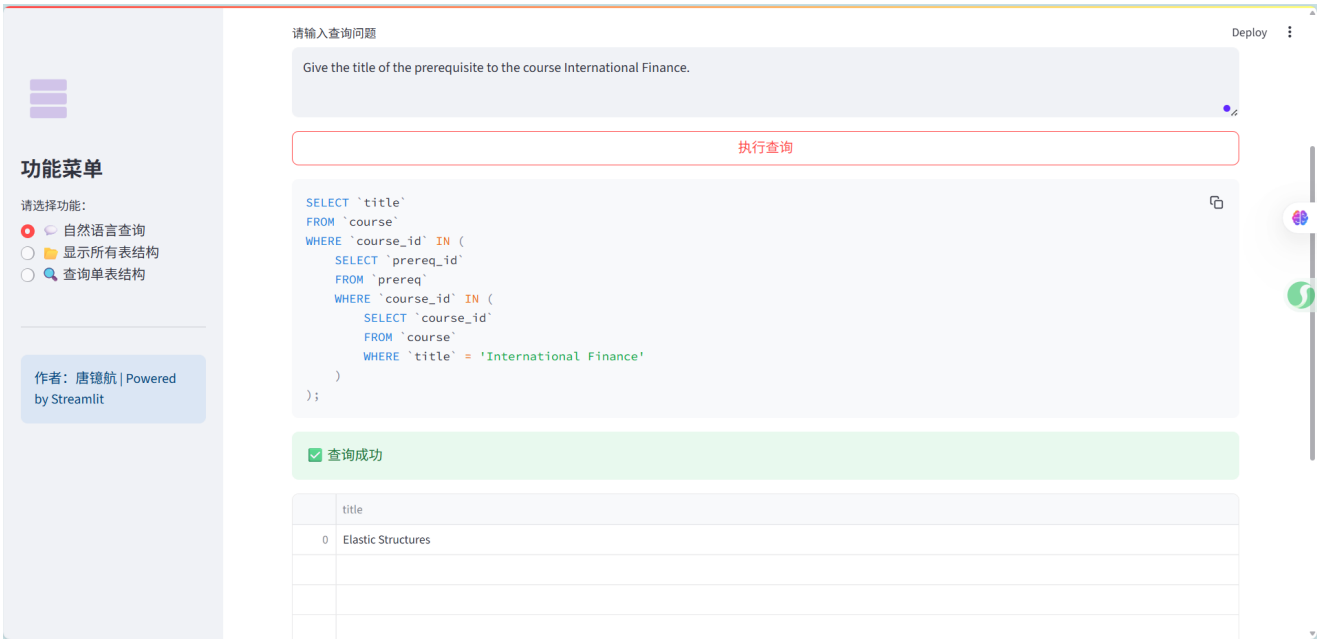


给出的SQL语句符合问题要求，结果正确

Question 6: What is the name of the department with the most credits? (hard)

给出的SQL语句符合问题要求，结果正确

Question 7: Give the title of the prerequisite to the course International Finance. (hard)



给出的SQL语句符合问题要求，结果正确

可以看到这一次测试的七个问题**全部正确**，正确率为**100%**
对比之前测试的七个问题中有**五个问题正确**，正确率为**71.4%**
正确率大幅提示

项目概述

代码结构

```
.
├── main.py                    # MCP服务以及查询控制模块，提供/schema和/query等HTTP
接口
```

```
├── cli_app_main.py          # 命令行主菜单交互入口
├── gui_app.py               # Streamlit可视化界面入口
├── tongyi_api_nl2sql.py     # 通义API自然语言转SQL模块
├── pyproject.toml           # Python依赖管理
├── logs/
│   └── query.log            # 查询日志
└── .venv/                   # 推荐的Python虚拟环境
```

主要功能

MCP 服务运行：下载并部署 alexcc4/mcp-mysql-server，连接 MySQL 实例|
自然语言转SQL： 集成通义大模型API，支持复杂查询意图的SQL自动生成
数据库结构浏览：支持全库和单表结构可视化
查询结果分页：CLI端自动分页，GUI端表格滚动
安全防护：只读SQL、敏感字段拦截、SQL注入检测
日志记录：所有查询均有日志可查
Prompt 模板优化：多轮提示结构 / 示例增强 Few-shot以及SQL 执行计划简化建议，提高生成 SQL 的准确率

存在的问题及解决方案

SQL 注入防御机制

一开始对根据对SQL注入防御机制的简单了解编写了注入防护的正则规则如下：

```
INJECTION_PATTERNS = [
    r"(--
|#|/\*|;|\bOR\b|\bAND\b|\bUNION\b|\bSLEEP\b|\bBENCHMARK\b|\bLOAD_FILE\b|\bIN
TO\b|\bOUTFILE\b)",
    r"(['\"]\s*\+|\+\s*['\"])",      # 字符串拼接
    r"\b1\s*=\s*1\b",                # 条件恒真
    r"\b0\s*=\s*0\b",                # 条件恒真
    r"\bselect\b.*\bselect\b",       # 嵌套子查询诱导
]
```

SQL检测逻辑如下：

```
def is_sql_injection(sql):
    sql_lower = sql.lower()
    for pattern in INJECTION_PATTERNS:
        if re.search(pattern, sql_lower):
            return True
    return False
```

但是在编写完这部分代码后对之前测试过的问题：Give the title of the prerequisite to the course International Finance.进行测试，结果显示：

```
请输入您的查询问题(exit 返回主菜单，quit 退出程序): Give the title of the prerequisite to the course International Finance.

生成的 SQL 语句：
SELECT `title` FROM `course` WHERE `course_id` = (SELECT `prereq_id` FROM `prereq` WHERE `course_id` = 'FIN-301')
查询失败：检测到疑似SQL注入攻击，已拦截。
```

观察发现因为前面的防护机制直接限制了所有的嵌套子查询，因此即使是合法的子查询也会被拦截。
为了实现更**精细的 SQL 注入防护机制**，同时**避免误拦合法子查询**，后面修改正则规则使得：

- 允许合法的子查询（如 `SELECT ... FROM ... WHERE ... IN (SELECT ...)`）
- 拦截明显的拼接攻击、盲注、批量执行、多语句
- 提升检测精度，减少误判

数据库连接失败

问题表现：
运行 CLI 或 MCP 服务时，报错 `connect() argument 2 must be str, not None` 或提示无法连接数据库。

原因分析：
未正确设置数据库连接环境变量，`.env` 文件缺失或变量名拼写错误。

解决方案：

- 在项目根目录下创建 `.env` 文件，正确填写 `DB_HOST`、`DB_USER`、`DB_PASSWORD`、`DB_NAME`、`DB_PORT` 等变量。
- 代码中增加了 `dotenv` 自动加载和环境变量检查，缺失时会有中文报错提示。

Prompt工程与SQL准确率

问题表现：
部分复杂自然语言问题生成的SQL不准确，如极值并列、子查询多行等。

原因分析：
Prompt工程不完善，缺少Chain-of-Thought、Few-shot等技术。

解决方案：

- 参考Prompt Engineering最佳实践，优化system prompt，增加思维链、示例、注意事项。
- 多轮测试和微调，显著提升SQL生成准确率。

实验小结

通过本次实验，我成功实现了一个基于 MCP 协议的自然语言数据库查询系统，涵盖了大模型 API 调用、SQL 转换控制、数据库查询执行、命令行与图形化交互界面、功能增强与安全控制等多个模块，全面锻炼了数据库编程与系统集成能力。

首先，在**系统搭建阶段**，我掌握了 MCP 服务器的本地部署流程，包括虚拟环境管理、依赖安装、环境变量配置与调试测试。这一过程不仅加深了我对服务端开发的理解，也提升了我解决环境兼容性与依赖冲突的能力。

在**自然语言转 SQL 模块**中，我通过调用通义千问大模型，设计了系统提示词与结构注入逻辑，实现了从用户自然语言到结构化 SQL 的转换功能。通过 Prompt 工程优化，显著提升了模型生成 SQL 的准确性和鲁棒性，解决了如多行子查询、并列最大值等复杂逻辑表达不准确的问题。

在**后端查询服务**开发中，我基于 Flask 构建了标准化 RESTful API，实现了数据库结构提取、查询执行、日志记录、分页返回等功能。同时，在 CLI 和 GUI 两个界面模块的构建中，我熟练运用了 `colorama`、`tabulate` 和 `streamlit` 等工具，增强了用户交互体验和系统实用性。

在**安全控制方面**，我实现了 SELECT 白名单限制、敏感字段过滤与 SQL 注入防御机制等策略，并通过正则表达式设计与调试，有效平衡了安全性与功能的合理开放，进一步增强了系统的实用性与健壮性。

整个实验过程中，我不仅解决了环境配置、API响应错误、Prompt效果不佳等多个技术难题，还在不断测试与迭代中构建了一个较为完整、可扩展、可视化的自然语言数据库查询系统。这不仅提升了我的实际开发能力和系统设计能力，也使我对大语言模型在数据库应用场景中的潜力有了更深入的认识。

综上，本次实验是一项综合性强、实践价值高的训练过程，对我今后的数据库系统开发与人工智能应用设计具有重要意义。