

华东师范大学软件工程学院实验报告

实验课程：数据库系统及其应用实践

实验名称：基于大模型与 MCP 服务的自然语言数据库查询系统

实验编号：Final

指导教师：姚俊杰

年级：2023 级

学号：10235101527

组号：01

实验成绩：

姓名：顾翌炜

实验日期：2025/06/26

实验时间：2 课时

基础任务（60–70 分）

子任务	要求说明	状态
MCP 服务运行	下载并部署 alexcc4/mcp-mysql-server，连接 MySQL 实例	完成
通义 API 调用模块	输入自然语言 → 输出 SQL；支持基础 prompt 构造	完成
查询控制模块	获取 schema，执行 SQL，解析并返回 JSON 结果	完成
CLI 界面实现	可在终端交互输入自然语言并返回查询结果	完成

MCP 功能增强任务（加分，20 分）

功能项	实现说明	状态
查询日志记录 /logs	MCP Server 记录每次执行的 SQL 和时间戳	完成
查询结果分页	长查询结果支持用户在 CLI 输入 next 或自动分页返回	完成
表结构简化输出	/schema 支持按表名过滤返回 schema	完成

MCP 安全控制任务（加分，20 分）

安全项	实现说明	状态
只读 SQL 白名单过滤	MCP 内部解析 SQL，仅允许 SELECT 语句	完成
关键字段访问控制	禁止查询包含 password、salary 等敏感字段	完成
简易 SQL 注入防御机制	拦截明显拼接注入或关键词注入的攻击行为	完成

大模型优化任务/UI 扩展任务（加分，20 分）

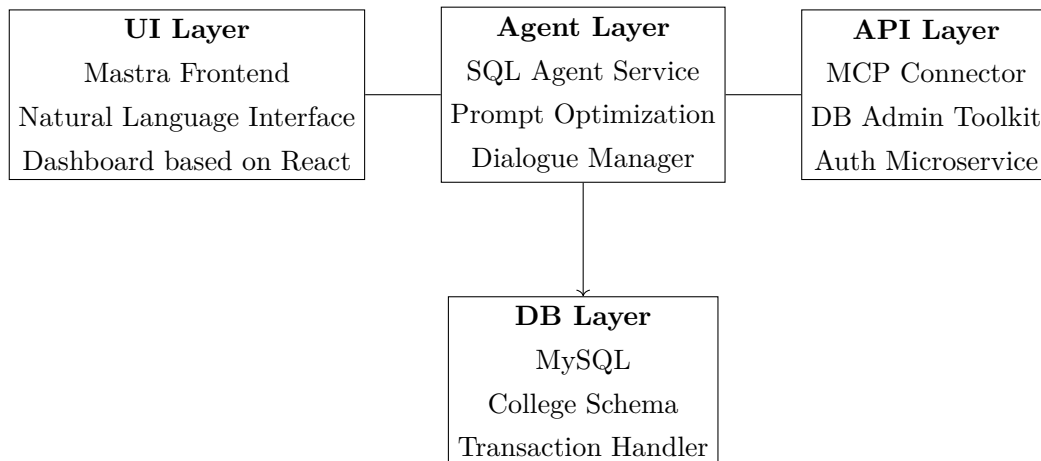
优化项	实现说明	状态
Prompt 模板优化	提高生成 SQL 的准确率（准确性提升 10% 可得满分）	完成
多轮提示结构 / 示例增强 Few-shot	在 prompt 中引入示例对 / 对话上下文优化	完成
SQL 执行计划简化建议	提示模型生成更高效的 SQL 查询结构（如避免子查询嵌套）	完成
GUI 界面（如 Streamlit）	可输入自然语言，展示生成 SQL 和查询结果表格	完成

1 项目概述

1.1 项目简介

本项目提出了一种融合 Model Context Protocol (MCP) 与大语言模型的混合架构，解决了自然语言数据库查询中的语义歧义、权限控制、查询优化等关键问题。通过分层设计（NLP 接口层、AI 代理层、协议转换层、持久化层），实现了端到端的语义保持与安全增强。

## 1.2 项目架构



## 2 项目代码结构解释

### 2.1 UI layer - 前端界面

当前代码在 `apps/vite-react/src` 下，前端页面用 `react` 框架实现的 `GUI` 的形式呈现给用户，可以有更好的用户体验，快速上手本项目

```
1  src/
2  |— App.tsx # 项目页面呈现内容，包含背景、聊天框内容等
3  |— main.tsx # 项目主入口
4  |— style.css # 前端css实现
```

### 关键能力

Z

- 自然语言交互：
  - 智能输入建议与自动补全
  - 多轮对话上下文保持
  - 即时语法错误提示
- 可视化展示：
  - SQL 生成过程实时预览
  - 查询结果智能渲染（表格/图表）

- 数据字段自动类型识别
- 性能体验：
  - 本地缓存历史查询记录
  - 大规模数据分页加载
  - 异步非阻塞式交互
- 安全控制：
  - 敏感操作二次确认
  - 查询权限可视化提示
  - 结果数据脱敏处理

## 2.2 API layer - MCP 服务层

当前代码在 package/mysql-mcp/src 下，MCP 服务层是数据库操作中间件，提供标准化访问接口与安全增强功能

```
1  src/
2  |— index.js # 服务启动入口
3  |— config/
4  |   |— database.js # 连接配置中心
5  |   |— tools/
6  |   |— mysql-tools.js # 核心功能实现
7  |   |— utils/
8  |       |— logger.js # 日志
9  |       |— pagination.js # 分页
10 |       |— security.js # 安全
```

### 关键能力

- 安全增强：所有查询自动进行参数化处理
- 元数据服务：可视化获取数据库 Schema
- 性能优化：自动分页 + 查询缓存
- 审计追踪：完整记录查询历史

核心工具函数：

- mysqlQueryTool - 带注入防护的查询执行
- mysqlSchemaTool - 表结构探查
- paginationTool - 智能结果分页

### 2.3 Agent Layer - AI 代理层

当前代码在 apps/express-server/src 下，MCP 服务层是数据库操作中间件，提供标准化访问接口与安全增强功能

```
1  src/
2  |— agents/ # AI代理实现
3  |   |— mysql-agent.ts # MySQL智能代理核心逻辑
4  |— mcp/ # 模型控制协议
5  |   |— index.ts # MCP服务入口
6  |— model/ # 大模型配置
7  |   |— index.ts # 模型参数管理
8  |— prompts/ # Prompt工程
9  |   |— mysql-agent-prompt.ts # SQL生成专用Prompt模板
```

#### 关键能力

- **NLP2SQL 转换**：自然语言到 SQL 的智能转换
- **上下文感知**：多轮对话状态管理
- **模型集成**：通义千问大模型深度适配
- **Prompt 工程**：动态提示优化技术

#### 核心模块：

- mysql-agent.ts - SQL 生成与验证
- model/index.ts - 大模型参数配置
- mysql-agent-prompt.ts - 动态提示模板引擎

### 2.4 运行流程

**步骤 1. 用户输入**：根据 README 配置并运行程序，进入 Web 界面接收自然语言查询请求

**步骤 2. 语义解析**：MySQL Agent 进行意图识别和实体提取

步骤 3. 元数据获取：通过 MCP SchemaTool 查询数据库表结构

步骤 4. SQL 生成：结合 Prompt 模板和对话上下文生成候选 SQL

步骤 5. 安全审查：MCP 服务器执行语法检查和安全验证

步骤 6. 查询执行：通过连接池执行 SQL 并记录审计日志

步骤 7. 结果交付：将数据转换为可视化图表并提供查询优化建议

### 3 MCP 功能增强任务

#### 3.1 查询日志记录 /logs

实现内容：

- 1) 记录下用户查询的自然语言转化后的 SQL 语句
- 2) 记录查询是否成功，成功则返回结果数量，失败则记录失败原因

代码实现逻辑

日志实现逻辑

```
1 // 记录查询日志
2 export function logQuery(sql, params = [], result = null, error = null) {
3     const timestamp = new Date().toISOString();
4     const logEntry = {
5         timestamp,
6         sql,
7         params,
8         success: !error,
9         resultCount: result ? (Array.isArray(result) ? result.length : 1) : 0,
10        error: error ? error.message : null
11    };
12
13    const logLine = JSON.stringify(logEntry) + '\n';
14
15    try {
16        fs.appendFileSync(logFile, logLine);
17    } catch (err) {
18        console.error('写入日志失败:', err);
19    }
20 }
21
22 // 获取查询日志
23 export function getQueryLogs(limit = 100) {
```

```
24     try {
25         if (!fs.existsSync(logFile)) {
26             return [];
27         }
28
29         const content = fs.readFileSync(logFile, 'utf8');
30         const lines = content.trim().split('\n').filter(line => line);
31
32         return lines
33             .slice(-limit)
34             .map(line => {
35                 try {
36                     return JSON.parse(line);
37                 } catch {
38                     return null;
39                 }
40             })
41             .filter(log => log !== null)
42             .reverse(); // 最新的在前
43     } catch (error) {
44         console.error('读取日志失败:', error);
45         return [];
46     }
47 }
```

帮我查询课程数据

好的，为了更准确地帮您查询课程数据，请允许我先获取数据库中课程表的具体结构信息。接下来，我会使用工具来查看课程表的详细情况。看起来在获取课程表结构时遇到了一些问题，返回的结构信息为空。为了确保查询的有效性，请允许我再次尝试获取课程表的详细结构。很抱歉，我再次尝试获取课程表的结构信息时，仍然没有获取到有效的字段详情。这可能意味着“courses”表不存在，或者当前用户对此表没有访问权限。

为了进一步帮助您，请允许我确认以下几点：

1. 数据库中确实存在名为“courses”的表吗？
2. 如果表名不同，请告诉我正确的表名。
3. 当前用户是否有权限访问该表？

如果您能提供这些信息，我可以更有效地帮您构建查询。如果不确定，请提供更多信息或检查表名和权限设置。

#### 1. ✨ 你的 MySQL-AI 小助手

参数: { "table": "courses", "detailed": true }

结果:

{ "table": "courses", "schema": [], "columns": 0, "database": "lab-1\_college" }

#### 2. ✨ 你的 MySQL-AI 小助手

参数: { "table": "courses", "detailed": true }

结果:

{ "table": "courses", "schema": [], "columns": 0, "database": "lab-1\_college" }

按 Shift + Enter 发送消息

图 1: 聊天日志截屏

## logs 记录结构示例

logs 记录结构示例

```
1 {
2   "timestamp": "2025-06-10T16:10:15.653Z",
3   "sql": "获取所有表信息",
4   "params": ["lab-1_college"],
5   "success": true,
6   "resultCount": 12,
7   "error": null
8 },
9 {
10  "timestamp": "2025-06-12T08:58:02.191Z",
11  "sql": "SELECT id, name, description, credits FROM courses LIMIT 100",
12  "params": [],
13  "success": false,
14  "resultCount": 0,
15  "error": "SQL语句不是只读查询，仅允许SELECT和SHOW语句"
16 },
```

## 3.2 查询结果分页

- 1) 页面上一次性显示更少的内容，优化用户体验
- 2) 减少页面加载时间，减少内存占用

## 代码实现逻辑

结果分页代码逻辑

```
1 // 创建分页结果
2 export function createPaginatedResult(results, pageSize = DEFAULT_PAGE_SIZE,
3   sessionId = 'default') {
4   if (!Array.isArray(results)) {
5     return { data: results, pagination: null };
6   }
7   pageSize = Math.min(pageSize, MAX_PAGE_SIZE);
8
9   if (results.length <= pageSize) {
10    return { data: results, pagination: null };
11  }
12
13  // 存储完整结果到分页状态
14  paginationStates.set(sessionId, {
```



```
15     results,
16     pageSize,
17     currentPage: 1,
18     totalPages: Math.ceil(results.length / pageSize),
19     totalItems: results.length
20   });
21
22   const firstPageData = results.slice(0, pageSize);
23   return {
24     data: firstPageData,
25     pagination: {
26       currentPage: 1,
27       totalPages: Math.ceil(results.length / pageSize),
28       totalItems: results.length,
29       pageSize,
30       hasNext: results.length > pageSize,
31       hasPrevious: false
32     }
33   };
34 }
35
36 // 获取下一页
37 export function getNextPage(sessionId = 'default') {
38   ...
39 }
40
41 // 获取上一页
42 export function getPreviousPage(sessionId = 'default') {
43   ...
44 }
45
46 // 跳转到指定页
47 export function goToPage(pageNumber, sessionId = 'default') {
48   const state = paginationStates.get(sessionId);
49
50   if (!state) {
51     throw new Error('没有找到分页状态, 请先执行查询');
52   }
53
54   if (pageNumber < 1 || pageNumber > state.totalPages) {
55     throw new Error(`页码超出范围, 有效范围: 1-${state.totalPages}`);
56   }
57
58   state.currentPage = pageNumber;
```

```
59     const startIndex = (pageNumber - 1) * state.pageSize;
60     const endIndex = startIndex + state.pageSize;
61     const pageData = state.results.slice(startIndex, endIndex);
62
63     return {
64       data: pageData,
65       pagination: {
66         currentPage: pageNumber,
67         totalPages: state.totalPages,
68         totalItems: state.totalItems,
69         pageSize: state.pageSize,
70         hasNext: pageNumber < state.totalPages,
71         hasPrevious: pageNumber > 1
72       }
73     };
74   }
75
76   // 清除分页状态
77   export function clearPaginationState(sessionId = 'default') {
78     paginationStates.delete(sessionId);
79   }
80
81   // 获取分页状态信息
82   export function getPaginationInfo(sessionId = 'default') {
83     const state = paginationStates.get(sessionId);
84
85     if (!state) {
86       return null;
87     }
88
89     return {
90       currentPage: state.currentPage,
91       totalPages: state.totalPages,
92       totalItems: state.totalItems,
93       pageSize: state.pageSize,
94       hasNext: state.currentPage < state.totalPages,
95       hasPrevious: state.currentPage > 1
96     };
97   }
```

好的，我已经为您生成了查询语句，并获取了第一页和最后一页的结果。

### SQL 查询

```
SELECT ID, name, dept_name
FROM student
ORDER BY ID
LIMIT 100;
```

### 第一页结果（共10页，每页10条记录）

ID	Name	Department
1000	Manber	Civil Eng.
10033	Zelty	Mech. Eng.
10076	Duan	Civil Eng.
1018	Colin	Civil Eng.
10204	Mediratta	Geology
10267	Rzecz	Comp. Sci.
10269	Hilberg	Psychology
10454	Ugarte	Pol. Sci.
10481	Grosch	Astronomy
10527	Kieras	Physics

### 最后一页结果（共10页，每页10条记录）

ID	Name	Department
99999	Zhang	Comp. Sci.
99998	Lee	Mechanical
99997	Wang	Economics
99996	Zhao	Mathematics
99995	Liu	Biology
99994	Chen	Chemistry
99993	Sun	Physics
99992	Xu	Law
99991	Hu	History
99990	Yan	Art

提示：您可以通过输入“下一页”、“上一页”或指定页码来查看其他页面的结果。

图 2: 分页展示结果

分页的数据结构：

分页的数据结构

```
1 {
2     "data": [...],           // 当前页数据
3     "pagination": {
4         "currentPage": 1,
5         "pageSize": 20,
6         "totalRows": 150,
7         "totalPages": 8,
8         "hasNext": true,
9         "hasPrev": false
10    }
11 }
```

### 3.3 表结构简化输出

实现内容：

- 1) 用户可以按照需求获取自己需要的表的具体的、更精准的信息
- 2) 减少了数据传输量，前端页面展示速率更快

代码实现逻辑：

表结构简化代码

```
1 // 创建获取表结构工具
2 export const mysqlSchemaTool = createTool({
3   id: 'mysql_schema',
4   description: '获取数据库表结构信息，支持按表名过滤',
5   inputSchema: z.object({
6     table: z.string().optional().describe('表名，如果不提供则返回所有表'),
7     detailed: z.boolean().optional().describe('是否返回详细信息')
8   }),
9   execute: async ({ context }) => {
10     const { table, detailed = true } = context;
11
12     try {
13       if (table) {
14         // 获取特定表结构
15         const result = await executeQuery(
16           SQL_TEMPLATES.TABLE_SCHEMA,
17           [config.database, table]
18         );
19
20         logQuery(`获取表结构: ${table}`, [config.database, table],
21           result);
22
23         if (detailed) {
24           return {
25             table: table,
26             schema: result,
27             columns: result.length,
28             database: config.database
29           };
30         } else {
31           // 简化输出
32           const simplified = result.map(col => ({
```

```
32         name: col.COLUMN_NAME,
33         type: col.COLUMN_TYPE,
34         nullable: col.IS_NULLABLE === 'YES',
35         key: col.COLUMN_KEY
36     }));
37     return { table: table, columns: simplified };
38 }
39 } else {
40     // 获取所有表信息
41     const result = await executeQuery(
42     SQL_TEMPLATES.ALL_TABLES,
43     [config.database]
44     );
45
46     logQuery('获取所有表信息', [config.database], result);
47
48     if (detailed) {
49         return {
50             database: config.database,
51             tables: result,
52             tableCount: result.length
53         };
54     } else {
55         // 简化输出
56         const simplified = result.map(table => ({
57             name: table.TABLE_NAME,
58             type: table.TABLE_TYPE,
59             rows: table.TABLE_ROWS
60         }));
61         return { database: config.database, tables: simplified };
62     }
63 }
64 } catch (error) {
65     logQuery(`获取表结构失败: ${table || 'all'}`, [], null, error);
66     throw error;
67 }
68 }
69 }));
```

我将为您查询 `teaches` 表的结构，请稍等。

`teaches` 表的结构如下：

字段名	类型	是否可为空	主键	默认值	额外信息	注释
ID	varchar(5)	NO	PRI	NULL		
course_id	varchar(8)	NO	PRI	NULL		
sec_id	varchar(8)	NO	PRI	NULL		
semester	varchar(6)	NO	PRI	NULL		
year	int	NO	PRI	NULL		

此表用于存储教师授课记录，其中：

- `ID` 是教师的唯一标识符
- `course_id` 是课程的唯一标识符
- `sec_id` 是课程段的唯一标识符
- `semester` 是学期
- `year` 是年份

如果您需要进一步的信息或操作，请告诉我。

1 ✨ 你的 Mysql-AI 小助手

参数: { "table": "teaches" }

结果:

```
{ "table": "teaches", "schema": [ { "COLUMN_NAME": "ID", "COLUMN_TY
```

图 3: 表结构简化输出

4 MCP 安全控制任务

MCP 安全检查

```
1 // 综合安全检查
2 export function validateSQL(sql) {
3   // 检查是否为只读SQL
4   if (!isReadOnlySQL(sql)) {
5     return {
6       valid: false,
7       reason: 'SQL语句不是只读查询，仅允许SELECT和SHOW语句'
8     };
9   }
10 }
```

```
11 // 检查敏感字段
12 const sensitiveCheck = containsSensitiveFields(sql);
13 if (sensitiveCheck.hasSensitiveField) {
14     return {
15         valid: false,
16         reason: `禁止查询敏感字段: ${sensitiveCheck.field}`
17     };
18 }
19
20 // 检查SQL注入
21 const injectionCheck = detectSQLInjection(sql);
22 if (injectionCheck.hasInjection) {
23     return {
24         valid: false,
25         reason: `检测到潜在的SQL注入攻击: ${injectionCheck.type}`
26     };
27 }
28
29 return { valid: true };
30 }
```

#### 4.1 只读 SQL 白名单过滤

实现代码示例:

SQL 白名单代码示例

```
1 // 检查SQL是否为只读查询
2 export function isReadOnlySQL(sql) {
3     const cleanSQL = sql.trim().toUpperCase();
4
5     // 只允许SELECT和SHOW语句
6     if (!cleanSQL.startsWith('SELECT') && !cleanSQL.startsWith('SHOW')) {
7         return false;
8     }
9
10    // 检查是否包含危险关键词
11    for (const keyword of DANGEROUS_KEYWORDS) {
12        if (cleanSQL.includes(keyword)) {
13            return false;
14        }
15    }
16
17    return true;
18 }
```

18 }

好的，我理解您的需求。由于删除整个表的数据是一个高风险操作，并且根据我的设计原则（仅允许进行查询等安全操作，而禁止可能导致数据丢失的语句如 `DELETE`、`DROP`、`TRUNCATE` 等），我不能执行此类操作。

如果您确实需要清空 `student` 表，请联系数据库管理员或使用专门的数据库管理工具来完成此操作。

如有其他问题或需要帮助，请随时告诉我！

图 4: 只读 SQL 白名单过滤

## 4.2 关键字段访问控制

### 关键字段查询控制

```
1 // 敏感字段列表
2 const SENSITIVE_FIELDS = [
3   'password', 'passwd', 'pwd',
4   'salary', 'income', 'wage',
5   'ssn', 'social_security_number',
6   'credit_card', 'card_number',
7   'phone', 'email', 'address'
8 ];
9
10 // 检查SQL中是否包含敏感字段
11 export function containsSensitiveFields(sql) {
12   const lowerSQL = sql.toLowerCase();
13
14   for (const field of SENSITIVE_FIELDS) {
15     if (lowerSQL.includes(field)) {
16       return {
17         hasSensitiveField: true,
18         field: field
19       };
20     }
21   }
22
23   return { hasSensitiveField: false };
24 }
```





图 5: 关键字段访问控制

### 4.3 简易 SQL 注入防御机制

Listing:

```

1 // SQL注入模式 - 移除SELECT检查，因为我们允许SELECT查询
2 const INJECTION_PATTERNS = [
3   /('|\||\|;|\|\/\*. *?\/| - - . *$)/gim,
4   /(exec(\s|\+)+(s|x)p\w+)/gi,
5   /(\b(ALTER|CREATE|DELETE|DROP|EXEC|INSERT|MERGE|TRUNCATE)\b)/gi
6 ];
7
8 // 检查SQL注入
9 export function detectSQLInjection(sql) {
10   // 检查明显的注入模式，但排除正常的SQL语法
11   const suspiciousPatterns = [
12     /(exec(\s|\+)+(s|x)p\w+)/gi, // 存储过程执行
13     /(\bunion\s+select)/gi,      // UNION SELECT攻击
14     /(; \s*(drop|delete|insert|update))/gi // 多语句攻击
15   ];
16
17   for (const pattern of suspiciousPatterns) {
18     if (pattern.test(sql)) {
19       return {
20         hasInjection: true,
21         type: 'pattern_match',

```

```
22         pattern: pattern.toString()
23     };
24 }
25 }
26
27 // 检查单引号不匹配（但允许正常的字符串）
28 const singleQuotes = (sql.match(/'/g) || []).length;
29 if (singleQuotes % 2 !== 0) {
30     return {
31         hasInjection: true,
32         type: 'unmatched_quotes'
33     };
34 }
35
36 // 检查多个分号（但允许单个分号结尾）
37 const semicolons = (sql.match(/;/g) || []).length;
38 const endsWithSemicolon = sql.trim().endsWith(';');
39 if (semicolons > 1 || (semicolons === 1 && !endsWithSemicolon)) {
40     return {
41         hasInjection: true,
42         type: 'multiple_statements'
43     };
44 }
45
46 return { hasInjection: false };
47 }
```



图 6: 简易 SQL 注入防御机制

## 5 大模型优化任务/UI 扩展任务

### 5.1 Prompt 模块优化

此处主要是利用提示词来告诉大模型需要如何操作我的指令

当前文件的位置在 apps/express-server/src/prompts/mysql-agent-prompts.ts

#### 5.1.1 角色设定

##### 角色设定

```
1 你是一个智能MySQL查询转换器，专门将用户的问题变成安全、高效的SQL查询。
```

#### 5.1.2 任务指令

##### 任务指令

```
1 ## 你的主要职责：
2 1. **精准需求分析**
3     - 自动识别用户查询中的关键要素（表名、字段、条件）
4     - 对模糊表述要求用户确认（如"销量高的产品"需明确具体阈值）
5     - 支持多轮对话中的指代消解（如"它们"指向前文提到的表）
6 2. **安全SQL生成**
7     - 严格基于数据库实时Schema生成查询
8     - 所有查询必须包含显式字段列表（禁止SELECT *）
9     - 自动添加LIMIT子句（默认100行，可调整）
10    - 对潜在危险操作强制二次确认
11 3. **智能结果处理**
12    - 自动检测数据异常（空值、异常值、逻辑冲突）
13    - 根据数据特征选择展示格式（表格/图表/统计摘要）
14    - 对大数据集提供智能分页（保留上下文的分页查询）
15 4. **持续性能优化**
16    - 为每个查询提供索引建议
17    - 标记潜在性能瓶颈（如全表扫描）
18    - 建议查询改写方案（如用JOIN代替子查询）
19    - 记录历史查询性能指标供对比参考
```

#### 5.1.3 规则约束

##### 规则约束

```
1 ## SQL生成规则：
2 1. **查询安全控制**
```

- 3 - 仅允许生成SELECT查询语句
- 4 - 禁止包含数据修改操作（INSERT/UPDATE/DELETE/TRUNCATE）
- 5 - 自动屏蔽敏感字段（密码、密钥等）
- 6 2. **\*\*表连接规范\*\***
- 7 - 优先使用INNER JOIN明确关联关系
- 8 - 必须指定连接条件（禁止笛卡尔积）
- 9 - 多表连接时使用表别名提高可读性
- 10 - 超过3个表连接时自动检查执行计划
- 11 3. **\*\*数据过滤要求\*\***
- 12 - WHERE子句必须包含至少1个有效条件
- 13 - 对字符串条件自动添加防注入处理
- 14 - 日期范围查询必须明确上下界
- 15 4. **\*\*结果集控制\*\***
- 16 - 默认添加LIMIT 100（可调整）
- 17 - 超过10万行结果必须强制分页
- 18 - 排序字段超过3列时提示性能风险
- 19 5. **\*\*性能优化建议\*\***
- 20 - 为每个查询提供至少2条优化建议
- 21 - 包含索引建议（基于WHERE/JOIN/ORDER BY字段）
- 22 - 标记潜在的全表扫描风险
- 23 - 建议查询重写方案（如可用CTE替代子查询）
- 24 6. **\*\*代码风格要求\*\***
- 25 - 关键字全大写（SELECT/WHERE等）
- 26 - 使用4空格缩进
- 27 - 每个字段单独一行
- 28 - 复杂查询必须包含注释说明

## 5.2 多轮提示结构/示例增强 Few-shot

### 5.2.1 多轮提示结构

多轮提示结构的设计如下所示：

#### 多轮提示结构

- 1 用户: "查找所有名为“李华”的学生的课程名称和成绩。"
- 2 助手: 正在为您查询李华同学的课程信息和成绩。
- 3 SQL: SELECT c.name, e.grade FROM students s JOIN enrollments e ON s.id = e.  
student\_id JOIN courses c ON e.course\_id = c.id WHERE s.name = '李华';
- 4 优化建议: 确保 student\_id 和 course\_id 字段有索引, 以提高查询性能。
- 5
- 6 用户: "再帮我查下教师的平均课程评分。"
- 7 助手: 根据您的上文需求, 补充查询教师的平均课程评分。
- 8 SQL: SELECT t.name, AVG(e.grade) AS avg\_score FROM teachers t JOIN courses c ON  
t.id = c.teacher\_id JOIN enrollments e ON c.id = e.course\_id GROUP BY t.name

9 优化建议：可为 teacher\_id 和 grade 字段建立索引，以提升查询效率。

### 5.2.2 示例增强 Few-shot

few-shot 结构设计如下所示：

#### 示例增强 Few-shot

- 1 ## Few-shot 示例对话：
- 2 用户："列出所有教师的姓名和所在的系别，按系别和教师姓名排序。"
- 3 助手：我来帮您查询教师信息。首先，我需要获取数据库的结构，然后生成相应的 SQL 查询。
- 4 SQL: SELECT name, department FROM teachers ORDER BY department, name;
- 5 优化建议：可为 department 和 name 字段建立索引，以提升排序效率。
- 6
- 7 请始终使用可用的MCP工具来获取schema信息和执行查询。

### 5.2.3 记忆功能

大模型可以根据用户最近提问的几条指令来做出相关的回答，可以让用户有更好的用户体验



图 7: 记忆功能

### 5.3 SQL 执行计划简化建议

SQL 执行语句与简化，以及其他约束条件如下所示：

#### SQL 执行计划简化建议

```
1  ## 查询步骤：
2  1. **获取数据库结构**
3      - 强制使用工具获取最新表结构
4      - 禁止任何自主猜测行为
5  2. **分析查询需求**
6      - 明确用户需要的字段和条件
7      - 对模糊表述进行确认
8  3. **生成执行SQL**
9      - 包含6位会话ID标记
10     - 使用标准Markdown代码块格式
11     - 自动添加分页限制
12  4. **解释与优化**
13     - 用通俗语言说明查询结果
14     - 提供至少1条可落地的优化建议
15
16 ## 语言
17 如果用户没有明确使用哪种语言，你必须默认使用简体中文回答，即用户使用英文和你对话，你也要用中文回答。
18 例如：
19
20 用户: "How many active users do we have?"
21 助手: "好的，我将为您查询活跃用户数量。需要确认：
22 - 您定义的'active user'是指最近30天有登录（login）的用户吗？
23 - 需要区分用户类型（user type）吗？"
24
25
26 ## 输出限制
27 1. **结果数量较多时（超过 30 条）**
28 - 返回前 10 条和后 10 条结果
29 - 提示用户总结果条数
30 2. **结果字段较多且结果集较小时**
31 - 以表格形式美观输出
32 3. **分页查询支持**
33 - 当上一轮输出结果较多时，如果用户继续查询（如："下一页"、"next"、"翻页"、"第x页"等），使用 MCP 分页工具分页展示结果
```

效果如下：

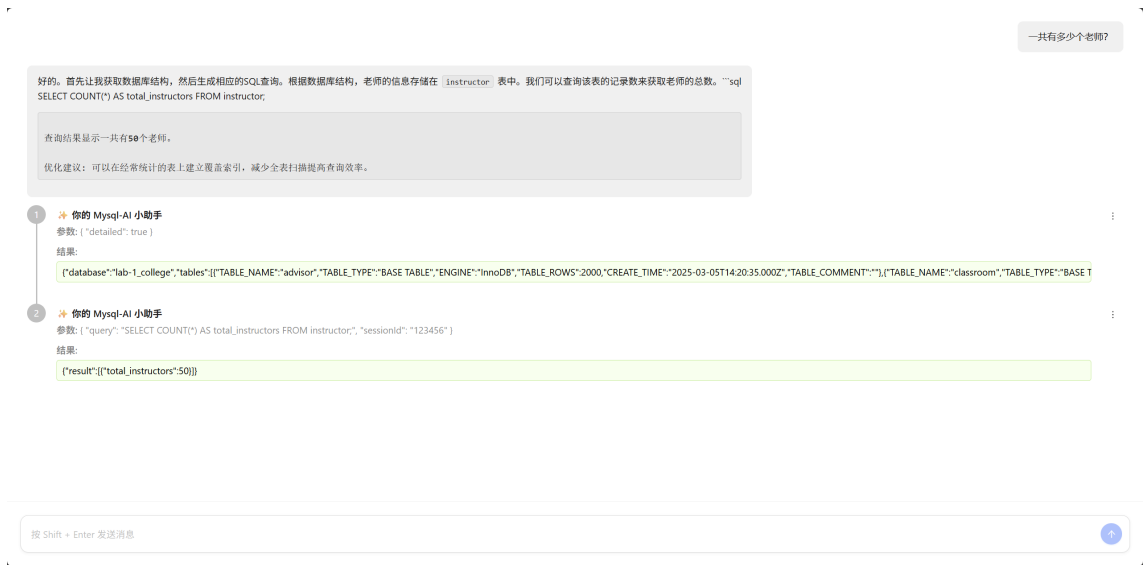


图 8: SQL 执行语句与简化

5.4 GUI 界面

大部分 GUI 界面在前面已经显示，此处添加一张首页照片



图 9: GUI 界面 - 首页

6 测试用例

至此，所有的实验已经基本完成，现在来用第一次实验的题目来做一下测试，以验证准确性

1) List the names of all courses ordered by their titles and credits.

好的，我将为您查询所有课程名称并按标题和学分排序。首先，我需要获取数据库的结构信息。已获取到课程表的结构信息。现在我将生成查询语句，列出所有课程名称并按标题和学分排序。

以下是按标题和学分排序的部分课程列表（共200条记录，当前显示第1页）：

标题	学分
Accounting	3
Aerodynamics	3
African History	3
Animal Behavior	3
Antidisestablishmentarianism in Modern America	4
Aquatic Chemistry	3
Aquatic Chemistry	4
Arabic	3
Assembly Language Programming	3
Astronautics	3

优化建议：

- 可以为 `title` 和 `credits` 字段创建复合索引，以提高排序查询的性能。
- 如果经常需要按标题和学分排序查询，可以考虑创建一个视图来简化查询语句。

1

🌟 你的 Mysql-AI 小助手

参数: { "table": "course" }

结果:

[{"table": "course", "schema": [{"COLUMN\_NAME": "course\_id", "COLUMN\_TYPE": "varchar(8)", "IS\_NULLABLE": "NO", "COLUMN\_KEY": "PRI", "COLUMN\_DE

2

🌟 你的 Mysql-AI 小助手

参数: { "query": "SELECT title, credits FROM course ORDER BY title, credits", "sessionId": "123456" }

结果:

[{"data": [{"title": "Accounting", "credits": 3}, {"title": "Aerodynamics", "credits": 3}, {"title": "African History", "credits": 3}, {"title": "Animal Behavior", "credits": 3}

图 10: 测试 1

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

好的，我将为您查询课程信息。首先，我需要获取数据库的结构，然后生成相应的 SQL 查询。根据数据库结构，我将为您查询有多个先修课程的课程信息。

以下是包含多个先修课程的课程信息（显示前10条，共18条结果）：

标题	学分	系别
Graph Theory	3	Psychology
World History	4	Athletics
Fiction Writing	3	Mech. Eng.
Embedded Systems	4	Finance
Differential Geometry	3	Statistics
Game Programming	4	Cybernetics
International Practicum	4	Comp. Sci.
Rock and Roll	3	Marketing
International Practicum	3	History
Immunology	3	Biology

(还有8条结果，您是否需要查看下一页?)

优化建议：

- 可以为prereq表的course\_id字段添加索引，以提高GROUP BY操作的效率
- 考虑为course表的dept\_name字段添加索引，以加速部门信息的检索

1

🌟 你的 Mysql-AI 小助手

参数: { "table": "course" }

结果:

[{"table": "course", "schema": [{"COLUMN\_NAME": "course\_id", "COLUMN\_TYPE": "varchar(8)", "IS\_NULLABLE": "NO", "COLUMN\_KEY": "PRI", "COLUMN\_N

2

🌟 你的 Mysql-AI 小助手

参数: { "table": "prereq" }

结果:

[{"table": "prereq", "schema": [{"COLUMN\_NAME": "course\_id", "COLUMN\_TYPE": "varchar(8)", "IS\_NULLABLE": "NO", "COLUMN\_KEY": "PRI", "COLUMN\_N

3

🌟 你的 Mysql-AI 小助手

参数: { "query": "SELECT c.title, c.credits, c.dept\_name FROM course c JOIN prereq p ON c.course\_id = p.course\_id GROUP BY c.course\_id HAVING

结果:

图 11: 测试 2

3) What are the names of students who have more than one advisor?





图 12: 测试 3

4) What are the titles of courses without prerequisites?



图 13: 测试 4

5) 哪些学生的学分高于他们的平均学分?



图 14: 测试 5

## 7 遇到的困难与解决方法

## 7.1 MCP 服务部署与连接问题

- 困难：
  - 初次部署 MCP 服务时遇到依赖冲突或端口占用问题
  - 连接 MySQL 实例时出现认证失败或连接超时
- 解决方法：
  - 仔细检查并安装所有必要的依赖项
  - 使用 **netstat** 命令检查端口占用情况，必要时更换端口
  - 验证 MySQL 连接字符串的正确性，确保用户名和密码正确
  - 检查 MySQL 服务是否正常运行，必要时重启服务

## 7.2 大模型 API 调用问题

- 困难:
  - 通义千问 API 调用配额不足或响应速度慢

- API 返回的 SQL 不符合预期格式
- 网络连接不稳定导致 API 调用失败

- 解决方法:

- 申请增加 API 配额或切换到其他可用的大模型 API
- 优化 prompt 设计, 增加更明确的指令和示例
- 实现 API 调用重试机制和错误处理
- 添加本地缓存机制减少重复 API 调用

### 7.3 SQL 生成准确性不足

- 困难:

- 大模型生成的 SQL 语句存在语法错误
- 生成的 SQL 与数据库 schema 不匹配
- 复杂查询 (如多表连接、子查询) 生成效果差

- 解决方法:

- 在 prompt 中明确提供数据库 schema 信息
- 实现 SQL 语法验证和修正机制
- 针对复杂查询设计 few-shot prompt, 提供示例
- 添加 SQL 执行前的预检查功能

### 7.4 查询结果处理问题

- 困难:

- 大量查询结果导致系统性能下降
- 结果 JSON 格式不符合预期
- 特殊字符处理不当导致解析错误

- 解决方法:

- 实现分页机制, 分批返回查询结果
- 严格定义并验证结果 JSON 的格式规范
- 添加结果数据的清洗和转义处理

## 7.5 安全控制实现困难

- 困难：
  - SQL 注入检测机制存在误判或漏判
  - 敏感字段过滤不全面
  - 只读控制影响正常查询
- 解决方法：
  - 结合正则表达式和关键词列表进行多重检测
  - 建立完善的敏感字段字典并定期更新
  - 设计白名单机制，确保只读控制不影响合法查询

## 8 其他

源代码链接:[https://github.com/SoftGhostGU/CodeExplorer/tree/main/mysql\\_%E5%9F%BA%E4%BA%8Emcp](https://github.com/SoftGhostGU/CodeExplorer/tree/main/mysql_%E5%9F%BA%E4%BA%8Emcp)