# 基于大模型的自然语言转SQL查询工具 - 概要设计文档

## 一、引言

### 1.1 文档目的

本概要设计文档详细描述了基于大模型的自然语言转SQL查询工具的系统架构、模块划分、接口设计、数据库设计等内容，为系统的详细设计和实现提供指导。

### 1.2 术语定义

| 术语 | 定义 |
| --- | --- |
| LLM | 大语言模型（Large Language Model），能够理解和生成自然语言的人工智能模型 |
| SQL | 结构化查询语言（Structured Query Language），用于管理关系型数据库 |
| E-R图 | 实体关系图（Entity-Relationship Diagram），用于描述数据库中实体之间的关系 |
| Streamlit | 一个用于构建数据应用的Python库 |
| SQLite | 一个轻量级的关系型数据库管理系统 |
| Ollama | 一个用于本地运行大语言模型的工具 |

## 二、系统架构设计

### 2.1 系统整体架构

本系统采用前后端分离的架构设计，前端使用Streamlit构建用户界面，后端使用Python实现核心业务逻辑。系统主要分为以下几个层次：

1. **前端层**：负责用户交互和结果展示，包括自然语言输入、SQL执行结果展示、E-R图可视化等
2. **服务层**：负责核心业务逻辑，包括自然语言转SQL、SQL执行、错误修正等
3. **数据层**：负责数据库管理，包括数据库初始化、SQL执行、数据存储等
4. **大模型层**：负责自然语言理解和SQL生成，支持多种大模型，如Ollama、OpenAI等

### 2.2 模块划分

系统主要分为以下几个模块：

| 模块 | 主要功能 | 所属层次 |
| --- | --- | --- |
| 前端应用 | 用户交互和结果展示 | 前端层 |
| SQL生成服务 | 自然语言转SQL | 服务层 |
| SQL执行服务 | SQL执行和错误修正 | 服务层 |
| E-R图可视化服务 | 生成数据库实体关系图 | 服务层 |
| 数据库管理 | 数据库初始化和SQL执行 | 数据层 |
| LLM客户端 | 大模型调用和管理 | 大模型层 |

## 2.3 核心业务流程

### 2.3.1 自然语言查询流程

1. 用户在前端输入自然语言查询
2. 前端应用将查询发送给SQL生成服务
3. SQL生成服务调用大模型生成SQL语句
4. SQL生成服务将生成的SQL发送给SQL执行服务
5. SQL执行服务执行SQL语句
6. 如果执行成功，返回结果给前端；如果执行失败，调用错误修正服务
7. 错误修正服务调用大模型修正SQL语句
8. 重复步骤5-7，直到SQL执行成功或达到最大重试次数
9. 前端应用展示最终结果

### 2.3.2 E-R图生成流程

1. 用户在前端选择E-R图可视化功能
2. 前端应用请求E-R图可视化服务
3. E-R图可视化服务获取数据库结构信息
4. E-R图可视化服务生成E-R图
5. E-R图可视化服务将E-R图返回给前端
6. 前端应用展示E-R图

# 三、模块详细设计

## 3.1 前端应用模块

### 3.1.1 功能描述

负责用户交互和结果展示，包括自然语言输入、SQL执行结果展示、E-R图可视化等。

### 3.1.2 页面设计

1. **首页**：系统标题、功能导航、关于系统
2. **自然语言查询页**：
    - 自然语言输入框
    - 最大重试次数设置
    - 执行按钮
    - 处理步骤展示
    - 最终结果展示
3. **E-R图可视化页**：
    - E-R图展示区
4. **数据库结构页**：
    - 表列表
    - 表结构详情
    - 示例数据展示

### 3.1.3 核心组件

| 组件 | 功能 | 实现技术 |
| --- | --- | --- |
| 侧边栏导航 | 功能切换 | Streamlit sidebar |
| 文本输入区 | 自然语言输入 | Streamlit text_area |
| 结果展示区 | 查询结果展示 | Streamlit dataframe |
| 代码展示区 | SQL语句展示 | Streamlit code |
| E-R图展示 | 实体关系图展示 | Streamlit components.v1.html |

## 3.2 SQL生成服务模块

### 3.2.1 功能描述

负责将自然语言转换为SQL查询语句，包括大模型调用、表结构格式化等功能。

### 3.2.2 核心类和方法

| 类/方法 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| SQLGeneratorService | SQL生成服务类 | - | - |
| _format_table_schema | 格式化表结构 | - | 格式化后的表结构字符串 |
| generate_sql | 生成SQL | natural_language: str | 包含SQL语句的字典 |
| fix_sql | 修复SQL | sql: str, error_message: str | 包含修复后SQL的字典 |
| execute_sql | 执行SQL | sql: str | 包含执行结果的字典 |
| get_database_info | 获取数据库信息 | - | 包含表名和示例数据的字典 |

### 3.2.3 表结构格式化示例

```
表名：publishers
列信息：
  - publisher_id INTEGER NOT NULL PRIMARY KEY
  - publisher_name TEXT NOT NULL
  - country TEXT NOT NULL

表名：books
列信息：
  - book_id INTEGER NOT NULL PRIMARY KEY
  - title TEXT NOT NULL
  - isbn TEXT NOT NULL
  - publisher_id INTEGER NULL
  - publication_year INTEGER NULL
  - price REAL NOT NULL
  - stock INTEGER NOT NULL
外键关系：
  - publisher_id REFERENCES publishers(publisher_id)
```

## 3.3 SQL执行服务模块

### 3.3.1 功能描述

负责执行SQL查询并处理错误修正，包括SQL执行、错误检测、自动修正等功能。

### 3.3.2 核心类和方法

| 类/方法 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| SQLExecutionService | SQL执行服务类 | - | - |
| execute_natural_language_query | 执行自然语言查询 | natural_language: str, max_retries: int | 包含最终结果的字典 |

### 3.3.3 错误修正流程

1. 执行SQL语句，捕获错误
2. 将错误信息和原始SQL发送给大模型
3. 大模型生成修正后的SQL
4. 执行修正后的SQL
5. 如果仍然失败，重复步骤2-4，直到达到最大重试次数

## 3.4 E-R图可视化服务模块

### 3.4.1 功能描述

负责生成数据库的实体关系图，包括表结构获取、E-R图生成等功能。

### 3.4.2 核心类和方法

| 类/方法 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| ERDiagramService | E-R图可视化服务类 | - | - |
| generate_er_diagram | 生成E-R图文件 | output_format: str, output_path: str | 包含生成结果的字典 |
| get_er_diagram_svg | 生成E-R图SVG字符串 | - | SVG格式的E-R图字符串 |

### 3.4.3 E-R图生成技术

使用Graphviz库生成E-R图，支持多种输出格式，如PNG、SVG、PDF等。E-R图包含以下信息：

1. 表名和列名
2. 数据类型
3. 主键和外键
4. 表之间的关系

## 3.5 数据库管理模块

### 3.5.1 功能描述

负责数据库管理，包括数据库初始化、SQL执行、数据存储等功能。

### 3.5.2 核心类和方法

| 类/方法 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| DatabaseManager | 数据库管理类 | - | - |
| init_database | 初始化数据库 | - | - |
| execute_query | 执行SQL查询 | query: str, params: tuple | 包含执行结果的字典 |
| get_table_schema | 获取表结构 | - | 包含表结构的字典 |
| get_table_names | 获取表名列表 | - | 表名列表 |
| get_sample_data | 获取示例数据 | table_name: str, limit: int | 包含示例数据的字典 |

### 3.5.3 数据库设计

数据库采用SQLite，包含以下表：

1. **publishers**：出版社表
2. **authors**：作者表
3. **categories**：图书分类表
4. **books**：图书表
5. **book_authors**：图书-作者关系表
6. **book_categories**：图书-分类关系表
7. **users**：用户表
8. **borrow_records**：借阅记录表

## 3.6 LLM客户端模块

### 3.6.1 功能描述

负责大模型调用和管理，支持多种大模型，如Ollama、OpenAI等。

### 3.6.2 核心类和方法

| 类/方法 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| LLMClient | LLM客户端抽象基类 | - | - |
| generate_sql | 生成SQL | natural_language: str, table_schema: str | 包含SQL语句的字典 |
| fix_sql | 修复SQL | sql: str, error_message: str, table_schema: str | 包含修复后SQL的字典 |
| OllamaClient | Ollama客户端实现 | model: str | - |
| LLMFactory | LLM客户端工厂类 | - | - |
| create_client | 创建LLM客户端 | model_name: str, api_key: str | LLMClient实例 |

### 3.6.3 大模型集成设计

采用工厂模式创建LLM客户端，支持多种大模型：

1. Ollama：本地运行的大模型
2. OpenAI：OpenAI API
3. DeepSeek：DeepSeek API
4. 其他大模型：可通过扩展LLMClient抽象类实现

# 四、接口设计

## 4.1 模块间接口

| 调用者 | 被调用者 | 接口 | 参数 | 返回值 |
| --- | --- | --- | --- | --- |
| 前端应用 | SQL生成服务 | generate_sql | natural_language: str | 包含SQL语句的字典 |
| 前端应用 | SQL执行服务 | execute_natural_language_query | natural_language: str, max_retries: int | 包含最终结果的字典 |
| 前端应用 | E-R图可视化服务 | get_er_diagram_svg | - | SVG格式的E-R图字符串 |
| 前端应用 | SQL生成服务 | get_database_info | - | 包含表名和示例数据的字典 |
| SQL执行服务 | SQL生成服务 | generate_sql | natural_language: str | 包含SQL语句的字典 |
| SQL执行服务 | SQL生成服务 | execute_sql | sql: str | 包含执行结果的字典 |
| SQL执行服务 | SQL生成服务 | fix_sql | sql: str, error_message: str | 包含修复后SQL的字典 |
| SQL生成服务 | LLM客户端 | generate_sql | natural_language: str, table_schema: str | 包含SQL语句的字典 |
| SQL生成服务 | LLM客户端 | fix_sql | sql: str, error_message: str, table_schema: str | 包含修复后SQL的字典 |
| SQL生成服务 | 数据库管理 | execute_query | query: str, params: tuple | 包含执行结果的字典 |
| SQL生成服务 | 数据库管理 | get_table_schema | - | 包含表结构的字典 |
| SQL生成服务 | 数据库管理 | get_table_names | - | 表名列表 |
| SQL生成服务 | 数据库管理 | get_sample_data | table_name: str, limit: int | 包含示例数据的字典 |
| E-R图可视化服务 | 数据库管理 | get_table_schema | - | 包含表结构的字典 |

## 4.2 外部接口

| 接口名称 | 功能 | 参数 | 返回值 |
| --- | --- | --- | --- |
| Ollama API | 调用Ollama模型 | model: str, prompt: str, options: dict | 包含模型响应的字典 |
| OpenAI API | 调用OpenAI模型 | model: str, messages: list, temperature: float | 包含模型响应的字典 |
| DeepSeek API | 调用DeepSeek模型 | model: str, messages: list, temperature: float | 包含模型响应的字典 |

# 五、数据设计

## 5.1 数据库表结构

### 5.1.1 publishers表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| publisher_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 出版社ID |
| publisher_name | TEXT | NOT NULL | 出版社名称 |
| country | TEXT | NOT NULL | 出版社国家 |

### 5.1.2 authors表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| author_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 作者ID |
| author_name | TEXT | NOT NULL | 作者名称 |
| birth_year | INTEGER | - | 作者出生年份 |

### 5.1.3 categories表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| category_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 分类ID |
| category_name | TEXT | NOT NULL | 分类名称 |

### 5.1.4 books表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| book_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 图书ID |
| title | TEXT | NOT NULL | 图书标题 |
| isbn | TEXT | NOT NULL UNIQUE | 图书ISBN |
| publisher_id | INTEGER | REFERENCES publishers(publisher_id) | 出版社ID |
| publication_year | INTEGER | - | 出版年份 |
| price | REAL | NOT NULL | 图书价格 |
| stock | INTEGER | NOT NULL DEFAULT 0 | 库存数量 |

### 5.1.5 book_authors表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| book_id | INTEGER | REFERENCES books(book_id) | 图书ID |
| author_id | INTEGER | REFERENCES authors(author_id) | 作者ID |
| PRIMARY KEY | - | (book_id, author_id) | 联合主键 |

### 5.1.6 book_categories表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| book_id | INTEGER | REFERENCES books(book_id) | 图书ID |
| category_id | INTEGER | REFERENCES categories(category_id) | 分类ID |
| PRIMARY KEY | - | (book_id, category_id) | 联合主键 |

### 5.1.7 users表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| user_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 用户ID |
| user_name | TEXT | NOT NULL | 用户名称 |
| email | TEXT | NOT NULL UNIQUE | 用户邮箱 |
| role | TEXT | NOT NULL DEFAULT 'reader' | 用户角色 |

### 5.1.8 borrow_records表

| 列名 | 数据类型 | 约束 | 描述 |
| --- | --- | --- | --- |
| record_id | INTEGER | PRIMARY KEY AUTOINCREMENT | 记录ID |
| user_id | INTEGER | REFERENCES users(user_id) | 用户ID |
| book_id | INTEGER | REFERENCES books(book_id) | 图书ID |
| borrow_date | DATE | NOT NULL | 借阅日期 |
| return_date | DATE | - | 归还日期 |

## 5.2 示例数据

系统初始化时会填充以下示例数据：

1. **publishers**：5条示例数据
2. **authors**：5条示例数据

3. **categories**：5条示例数据
4. **books**：5条示例数据
5. **book_authors**：7条示例数据
6. **book_categories**：6条示例数据
7. **users**：3条示例数据
8. **borrow_records**：4条示例数据

# 六、系统部署与运行

## 6.1 部署环境

| 环境 | 要求 |
| --- | --- |
| 操作系统 | Windows 10/11，macOS 10.15+ |
| Python版本 | Python 3.10+ |
| 内存 | 8GB+ |
| 磁盘空间 | 10GB+ |

## 6.2 依赖安装

```
pip install -r requirements.txt
```

## 6.3 系统运行

```
python app.py
```

系统将启动Streamlit应用，可通过浏览器访问http://localhost:8504使用系统功能。

## 6.4 配置说明

系统配置通过.env文件进行管理，主要配置项包括：

| 配置项 | 描述 | 默认值 |
| --- | --- | --- |
| OPENAI_API_KEY | OpenAI API密钥 | 无 |
| DEEPSEEK_API_KEY | DeepSeek API密钥 | 无 |
| DEFAULT_MODEL | 默认使用的大模型 | llama3:8b |
| DATABASE_PATH | 数据库文件路径 | ./src/data/library.db |

# 七、测试计划

## 7.1 功能测试

| 测试项 | 测试内容 | 预期结果 |
| --- | --- | --- |
| 自然语言转SQL | 输入简单自然语言查询，检查生成的SQL是否正确 | 生成的SQL语法正确，能够执行 |
| SQL执行 | 执行生成的SQL，检查结果是否正确 | 执行成功，结果正确 |
| 错误修正 | 模拟SQL错误，检查系统是否能够自动修正 | 系统能够自动修正SQL，执行成功 |
| E-R图可视化 | 检查E-R图是否能够正确生成 | E-R图清晰展示数据库关系 |
| 数据库结构查看 | 检查数据库结构是否能够正确展示 | 正确展示表结构和示例数据 |

## 7.2 性能测试

| 测试项 | 测试内容 | 预期结果 |
| --- | --- | --- |
| 响应时间 | 测试自然语言转SQL的响应时间 | 响应时间不超过5秒 |
| 并发处理 | 测试系统的并发处理能力 | 支持10个并发用户 |

## 7.3 兼容性测试

| 测试项 | 测试内容 | 预期结果 |
| --- | --- | --- |
| 浏览器兼容性 | 在不同浏览器中测试系统功能 | 系统在主流浏览器中正常运行 |
| 操作系统兼容性 | 在Windows和Mac系统中测试系统功能 | 系统在不同操作系统中正常运行 |

# 八、总结

本概要设计文档详细描述了基于大模型的自然语言转SQL查询工具的系统架构、模块划分、接口设计、数据库设计等内容。该文档为系统的详细设计和实现提供了指导，确保系统能够按照预期目标完成。

系统采用模块化设计，具有良好的可扩展性和可维护性，支持多种大模型和数据库类型。系统的核心功能包括自然语言转SQL、SQL执行、自动错误修正和E-R图可视化等，能够为用户提供便捷、高效的数据查询体验。

通过本设计，系统将实现以下目标：

1. 降低用户使用数据库的门槛，支持自然语言查询
2. 提高数据查询的效率，自动生成和修正SQL
3. 增强系统的易用性，提供直观的用户界面和结果展示
4. 支持多种大模型，提高系统的灵活性和可扩展性
5. 提供E-R图可视化功能，帮助用户理解数据库结构

本设计符合大厂项目最佳实践，采用了模块化设计、工厂模式、抽象基类等设计模式，确保系统具有良好的可扩展性、可维护性和可测试性。