

车辆租赁管理系统实验报告

林杰泓 22336137

刘艺凡 22336162

2024 年 12 月 26 日

目录

1 实验题目	3
2 需求分析	3
2.1 功能需求	3
2.2 非功能需求	3
3 系统设计	3
3.1 系统结构图	3
3.2 系统功能模块图	4
3.3 E-R 图	5
3.3.1 实体及属性	5
3.3.2 实体之间的关系	6
3.4 数据库模式	7
3.4.1 表设计	7
3.4.2 数据库模式特点	8
3.5 安全性与完备性	8
3.5.1 用户认证与权限管理	8
3.5.2 数据完整性约束	8
3.5.3 数据保护与加密	9
3.5.4 日志记录与审计	9
3.5.5 最小权限原则	11
4 最终效果展示	11

1 实验题目

设计一个车辆租赁管理系统，包括车辆信息管理、租赁管理、客户管理等功能。车辆信息管理负责车辆信息的添加、修改和查询；租赁管理负责租赁信息的录入、修改和查询；客户管理负责客户信息的添加、修改和查询。

2 需求分析

2.1 功能需求

- 车辆信息管理：包括车辆的添加、修改、查询。每辆车有编号、品牌、型号、车牌号、租金等信息。
- 租赁管理：包括租赁记录的管理，租赁客户、租赁车辆等。
- 客户管理：管理客户信息，包含客户 ID、姓名、联系方式等。

2.2 非功能需求

- 安全性：对用户的权限进行控制，确保只有管理员可以进行修改操作。
- 系统响应时间：保证系统能快速响应用户操作，尤其是查询操作。
- 界面友好性：设计直观的用户界面，保证用户能够方便地操作和管理信息。

3 系统设计

3.1 系统结构图

车辆租赁管理系统采用分层架构设计，主要分为表现层、业务逻辑层和数据访问层，各层次的功能和作用如下：

- 表现层（前端）：
表现层是系统与用户交互的部分，负责用户界面的显示和操作处理，包括车辆信息的查询、客户信息录入、租赁信息的修改等功能。具体技术采用 HTML、CSS 等前端技术，确保界面美观和用户体验友好。
- 业务逻辑层（后端）：
业务逻辑层位于表现层与数据访问层之间，是系统功能实现的核心部分，主要负责接收前端请求、处理业务逻辑并与数据库交互。车辆管理、租赁管理和客户管理的功能均在该层实现。本系统采用 Django 框架开发业务逻辑层，支持高效的请求响应和模块化开发。

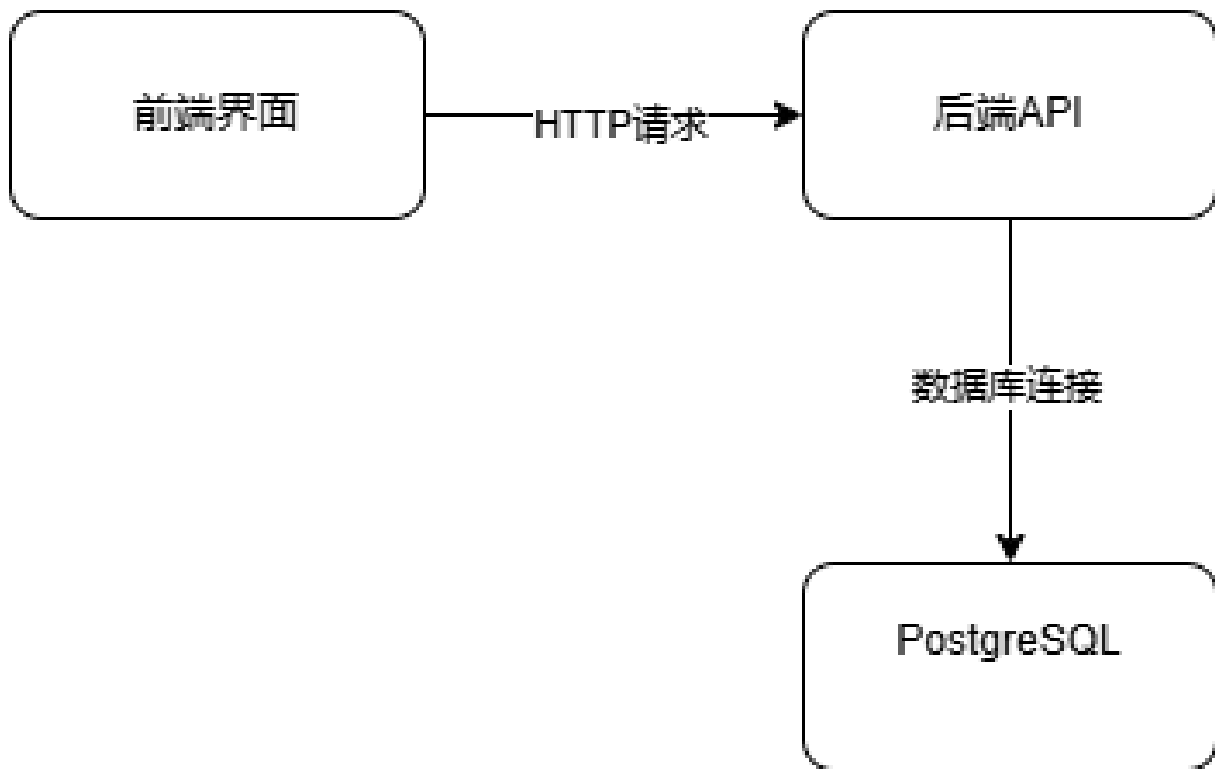


图 1: 系统结构图

- 数据访问层（数据库）：

数据访问层是系统的数据存储和管理部分，负责与数据库交互，执行 SQL 查询操作以实现数据的增删改查。本系统选用 PostgreSQL 作为数据库管理系统，设计了满足第三范式（3NF）的数据库结构，确保数据存储的规范性和完整性。

各层通过接口进行数据交互：表现层将用户请求发送给业务逻辑层，业务逻辑层根据功能需求调用数据访问层与数据库交互，最终返回结果至表现层供用户查看和操作。这种分层设计提升了系统的可维护性和扩展性。

针对需求分析，我们得到了系统结构图，如图1所示。

3.2 系统功能模块图

车辆租赁管理系统的功能模块图以系统需求为基础，分为三个主要功能模块：车辆信息管理模块、租赁管理模块和客户信息管理模块。各模块的功能和相互关系描述如下：

- 车辆信息管理模块：实现车辆信息的添加、修改等功能。
添加功能用于录入车辆基本信息，如车辆编号、车型、租赁价格等。
修改功能用于更新车辆状态或属性，如车辆的租赁状态、等。
- 租赁管理模块：实现租赁交易的管理，包括租赁信息的录入、修改和查询。
录入功能记录租赁交易信息，如租赁车辆、客户、起始日期、结束日期及费用等。

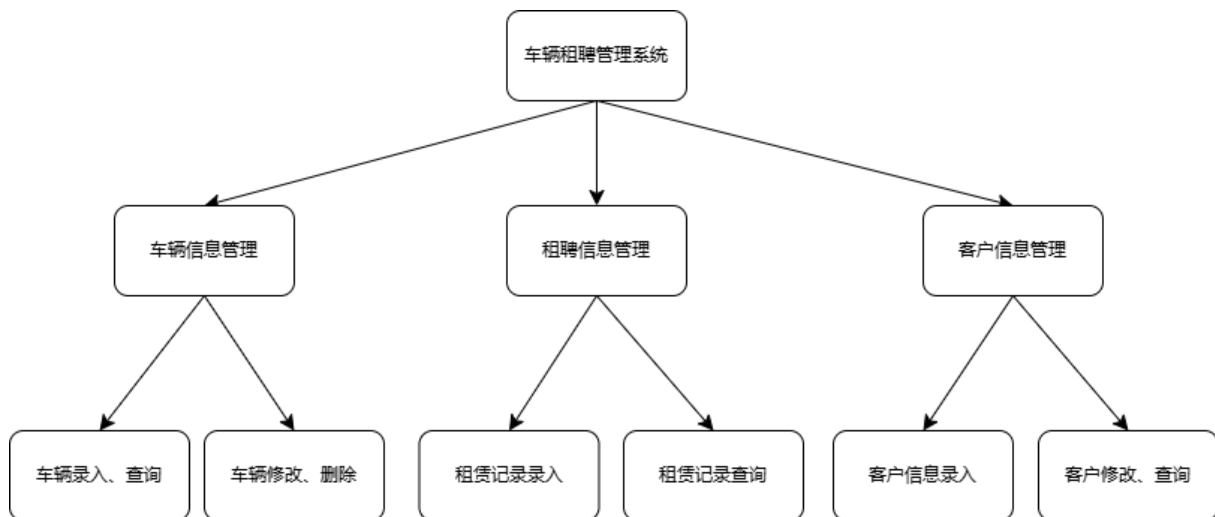


图 2: 系统功能模块图

修改功能允许调整租赁的相关信息，如租赁时间或费用的变更。

查询功能支持客户查找租赁记录，方便查看历史交易。

- 客户信息管理模块：实现客户信息的管理，包括添加、修改和查询功能。
添加功能用于录入新客户的信息，如姓名、联系方式、身份证号等。
修改功能用于更新客户信息，如更改联系方式或地址。
查询功能支持按客户姓名或其他条件检索客户信息，便于用户管理客户数据。

针对需求分析，我们得到了系统的功能模块图，如图2所示。

3.3 E-R 图

车辆租赁管理系统的 E-R 图描述了系统中的主要实体及其之间的关系，反映了系统的数据结构和逻辑关系。根据系统需求分析，设计了以下实体及其属性：

3.3.1 实体及属性

- 客户（Customer）属性：
用户（User）：与 Django 的内置用户模型进行一对一关联，扩展用户信息。
ID（主键）：客户唯一标识符。
姓名（name）：客户姓名。
联系方式（contact）：客户的联系方式。
- 车辆（Vehicle）属性：
型号（Model，主键）：车辆的唯一标识。
价格（Price）：车辆的租赁价格。

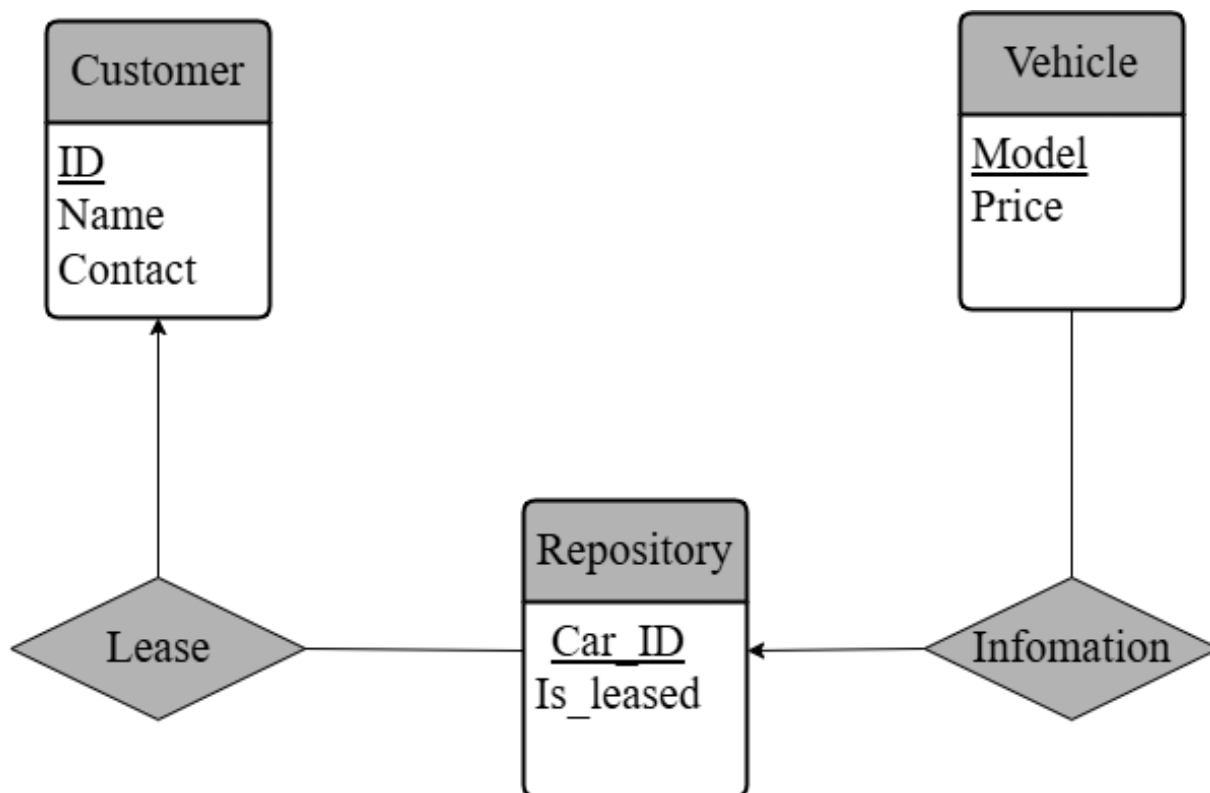


图 3: E-R 图

- 车辆库 (Repository) 属性:
车辆 ID (Car_ID, 主键): 标识库中每辆车的唯一编号。
是否租赁 (Is_leased): 表示车辆是否已被租赁。
- 车辆信息 (Info) 属性:
车辆 ID (Car_ID, 外键): 引用车辆库中的车辆 ID。
型号 (Model, 外键): 引用车辆表中的型号信息。
- 租赁记录 (Lease) 属性:
车辆 ID (Car_ID, 外键): 关联车辆库中的车辆。
客户 ID (ID, 外键): 关联客户表中的客户。

3.3.2 实体之间的关系

- 客户与租赁记录: 一个客户可以进行多次租赁记录 (一对多)。
- 车辆与车辆库: 车辆库中的每辆车属于一个车辆型号 (多对一)。
- 车辆库与租赁记录: 每辆车在租赁记录中最多出现一次 (一对一)。
- 车辆库与车辆信息: 每辆车在车辆信息表中有对应的详细记录 (一对一)。

针对需求分析，画出 E-R 图表示的概念模型，如图3所示。

3.4 数据库模式

根据车辆租赁管理系统的需求和 E-R 图，将概念模型转换为关系模型，设计出满足第三范式（3NF）的数据库模式。具体如下：

3.4.1 表设计

表 1: 客户表（Customer）

字段名	数据类型	约束	说明
ID	VARCHAR(10)	PRIMARY KEY	客户唯一标识符
user_id	INTEGER	UNIQUE, NOT NULL	关联 Django 用户表
name	VARCHAR(50)	NOT NULL	客户姓名
contact	VARCHAR(15)	NOT NULL	客户联系方式

表 2: 车辆表（Vehicle）

字段名	数据类型	约束	说明
Model	VARCHAR(50)	PRIMARY KEY	车辆型号
Price	INTEGER	NOT NULL	租赁价格

表 3: 车辆库表（Repository）

字段名	数据类型	约束	说明
Car_ID	VARCHAR(10)	PRIMARY KEY	车辆唯一标识符
Is_leased	BOOLEAN	DEFAULT FALSE	是否被租赁

表 4: 车辆信息表（Info）

字段名	数据类型	约束	说明
Car_ID	VARCHAR(10)	FOREIGN KEY (Repository.Car_ID), NOT NULL	关联车辆库表
Model	VARCHAR(50)	FOREIGN KEY (Vehicle.Model), NOT NULL	关联车辆型号

3.4.2 数据库模式特点

1. 满足第三范式（3NF）

- 消除冗余：每个表只存储一种实体的属性，避免数据冗余。

表 5: 租赁记录表 (Lease)

字段名	数据类型	约束	说明
ID	VARCHAR(10)	FOREIGN KEY (Customer.ID), NOT NULL	关联客户表
Car_ID	VARCHAR(10)	FOREIGN KEY (Repository.Car_ID), NOT NULL	关联车辆库表

- 数据依赖明确：所有非主键字段完全依赖于主键。

2. 完整性约束

- 主键约束：每个表都有明确的主键，保证数据唯一性。
- 外键约束：通过外键建立表间关系，保证数据一致性。

3. 扩展性

- 模块化表设计便于功能扩展，如增加车辆维修管理、客户等级管理等功能。

3.5 安全性与完备性

为了确保车辆租赁管理系统的数据库安全性与完备性，采取了以下设计和实现措施：

3.5.1 用户认证与权限管理

- **用户认证**：通过 Django 内置的用户认证系统（Authentication System），对系统用户进行登录验证，确保只有授权用户能够访问系统。
- **权限控制**：基于角色分配权限，不同用户（如管理员和普通用户）拥有不同的数据库访问权限。例如：
 - 管理员：拥有添加、修改、删除数据的权限。
 - 普通用户：仅能查看车辆信息和提交租赁请求。

3.5.2 数据完整性约束

- **主键约束**：每张表都有主键，确保每条记录具有唯一标识符。
- **外键约束**：外键关系保证了数据的一致性，例如租赁记录表中的车辆 ID 和客户 ID 必须合法且存在。
- **非空约束**：关键字段（如客户姓名、联系方式等）设置为非空，确保数据完整。
- **默认值约束**：为布尔字段（如车辆是否租赁）设置默认值，减少人为错误。

3.5.3 数据保护与加密

- **敏感数据加密：**对于用户的敏感信息（如密码），使用 Django 提供的密码哈希功能进行加密存储。

Django 的 User 模型已经内置了密码哈希功能,因此只需要使用 `create_user` 或 `set_password` 方法来处理用户密码。下面是我们实现的思路：

```
1      # 创建用户并加密密码
2      user = User.objects.create_user(username=username, password
                                     =password)
```

3.5.4 日志记录与审计

- 系统记录所有数据库操作的日志，包括数据添加、修改、删除等关键操作。
- 日志可用于追踪用户操作行为，及时发现和响应潜在的安全威胁。

```
1      LOGGING = {
2          'version': 1,
3          'disable_existing_loggers': False,
4          'formatters': {
5              'verbose': {
6                  'format': '{levelname} {asctime} {module} {message}',
7                      'style': '{',
8              },
9              'simple': {
10                 'format': '{levelname} {message}',
11                 'style': '{',
12             },
13         },
14         'handlers': {
15             'console': {
16                 'level': 'DEBUG',
17                 'class': 'logging.StreamHandler',
18                 'formatter': 'simple',
19             },
20             'file': {
21                 'level': 'INFO',
```

```

22         'class': 'logging.FileHandler',
23         'filename': 'myapp.log',
24         'formatter': 'verbose',
25     },
26 },
27 'loggers': {
28     'django': {
29         'handlers': ['console'],
30         'level': 'DEBUG',
31         'propagate': True,
32     },
33     'myapp': {
34         'handlers': ['console', 'file'],
35         'level': 'DEBUG',
36         'propagate': True,
37     },
38 },
39 }

```

在 `views.py` 中，我们可以记录用户操作日志，如下所示：

```

1  if User.objects.filter(username=username).exists():
2      logger.warning(f"Username_{username}_already_exists.
3      messages.error(request, 'Username_already_exists._
4      Please_choose_another_one.')
5      return render(request, 'management/register.html')

```

然后我们操作后就可以在 `myapp.log` 文件中看到相关日志。

```

1  WARNING 2024-12-25 10:36:47,223 views Username demo already
2  exists.
3  INFO 2024-12-25 10:41:41,707 views User demo logged in
4  successfully.
5  INFO 2024-12-25 10:41:43,616 views User demo returned
6  vehicle with ID XPENG001.
7  INFO 2024-12-25 10:41:54,603 views User demo returned
8  vehicle with ID XPENG001.

```

3.5.5 最小权限原则

- 数据库账户的权限设置遵循最小权限原则，只赋予完成任务所需的最低权限，避免权限过高导致的安全隐患。

通过以上安全性设计，系统能够有效保护数据免受未授权访问、篡改或丢失，确保数据库的完整性、保密性和可用性。

4 最终效果展示