北京交通大學
BEIJING JIAOTONG UNIVERSITY

航空公司乘客管理

系统详细设计说明书

学期：2024-2025 第二学期

编制日期：2025 年 06 月 04 日

编制人：江家玮

学号：22281188

班级：计科 2204

目录

1. 系统功能概述

航空公司乘客管理系统是一个核心目标为管理航空公司乘客信息的应用程序。其主要功能涵盖了乘客信息的录入、查询、修改和删除操作，同时还包括了与乘客紧密相关的航班预订信息的管理。该系统通过后端应用程序接口（API）与数据库进行交互，从而实现了对乘客数据、航班数据、预订数据等各类信息的持久化存储和高效管理。除此之外，系统还运用了存储过程和触发器来优化数据库层面的操作效率和业务逻辑的自动化处理，并且针对数据库并发操作可能引发的数据不一致性问题，系统也进行了相应的模拟分析和验证。

2. 系统功能模块结构

系统整体架构主要由三个核心部分组成：前端用户界面、WEB 服务端应用程序以及后端数据库。

2.1 前端功能 (基于 passengers.html)

前端为用户提供了一个直观易用的图形化界面，使得用户可以方便地与乘客管理系统进行交互。其主要功能具体体现在以下几个方面：首先，**乘客列表展示**功能能够以清晰的表格形式，将系统中所有乘客的关键信息，例如 ID、姓名、身份证号、电话号码、电子邮箱以及常旅客号码等，完整地呈现给用户。其次，**添加新乘客**功能通过一个模态对话框表单，允许用户便捷地输入新乘客的各项详细信息，包括姓名、身份证号、电话、邮箱和常旅客号，并将这些信息提交至后端服务进行保存处理。再次，**编辑乘客信息**功能允许用户从乘客列表中选择任意一位乘客，并在弹出的模态框表单中对其现有信息进行修改，修改完成后提交更新至数据库。此外，**删除乘客信息**功能为每条乘客记录提供了删除操作按钮，用户在执行删除前会收到确认提示，以防止误操作。系统还具备**实时反馈**机制，针对用户的各项操作（如添加成功、编辑失败、删除成功等），都会给出相应的即时提示信息，增强了用户体验。在数据加载过程中，系统会显示**加载指示**，明确告知用户当前的数据处理状态。最后，前端还包含

了**输入验证**逻辑，会对用户在表单中输入的数据（例如身份证号码的格式、必填项是否为空等）进行初步校验。入包括：

- **RESTful API 接口：**
  - ✓ WEB 服务端采用 Python Flask 框架进行构建，其核心职责是处理来自前端的各类请求、执行相应的业务逻辑，并负责与后端数据库进行所有的数据交互。其主要功能模块可以细分为：首先，提供了一系列 **RESTful API 接口**，这些接口又可以分为两类：一类是**乘客管理 API (直接 SQL 操作)**，包括 GET /api/passengers 用于获取所有乘客列表，GET /api/passengers/<passenger_id> 用于根据乘客 ID 获取单个乘客的详细信息，POST /api/passengers 用于添加新的乘客记录，PUT /api/passengers/<passenger_id> 用于更新指定 ID 乘客的信息，以及 DELETE /api/passengers/<passenger_id> 用于删除指定 ID 的乘客。另一类是**乘客管理 API (通过存储过程操作)**，例如 GET /api/sp/passengers/<passenger_id>/details 通过调用 GetPassengerAndBookings 存储过程来获取乘客及其相关的预订详情，POST /api/sp/passengers 通过 AddPassengerViaSP 存储过程添加新乘客，DELETE /api/sp/passengers/<passenger_id> 通过 DeletePassengerViaSP 存储过程删除乘客，以及 PUT /api/sp/passengers/<passenger_id>/email 通过 UpdatePassengerEmailViaSP 存储过程来更新乘客的邮箱信息。其次，服务端包含了**数据库连接管理**机制，提供了获取和管理数据库连接的辅助函数，确保数据库连接的有效和高效使用。再次，服务端实现了完善的**错误处理与响应**逻辑，能够捕获并处理数据库操作过程中可能发生的错误（例如身份证号重复、数据库连接失败等），并以统一的 JSON 格式将错误信息或操作结果响应给前端。最后，服务端还承担了 **HTML 页面服务**的功能，例如通过 GET / 路由提供系统主要的乘客管理 HTML 页面 (passengers.html) 给用户浏览器。/据的存储、管理和一致性维护。

- ✓ **数据表**：定义了系统的核心数据实体，如乘客、航班、预订等（详见 4.1 节）。

- **存储过程：**

  - ✓ GetPassengerAndBookings(IN p_passenger_id INT)：根据乘客 ID 查询乘客基本信息及其所有相关的预订记录（多表连接查询）。
  - ✓ AddPassengerViaSP(IN p_name VARCHAR(100), ..., OUT p_new_passenger_id INT)：向 Passenger 表中插入一条新的乘客记录，并返回新插入乘客的 passenger_id。
  - ✓ DeletePassengerViaSP(IN p_passenger_id INT)：根据乘客 ID 从 Passenger 表中删除对应的乘客记录。
  - ✓ UpdatePassengerEmailViaSP(IN p_passenger_id INT, IN p_new_email VARCHAR(100))：根据乘客 ID 和新的电子邮件地址，更新该乘客的电子邮件信息。

- **触发器：**

  - ✓ AfterBookingInsert (AFTER INSERT ON Booking)：当向 Booking 表成功插入一条新的预订记录后，自动将对应航班 (Flight 表) 的 booked_seats (已预订座位数) 加 1。
  - ✓ AfterPassengerEmailUpdate (AFTER UPDATE ON Passenger)：当 Passenger 表中某条记录的 email 字段被更新后（且新旧值不同），自动将旧邮箱、新邮箱及变更时间等信息记录到 PassengerEmailAudit 审计表中。
  - ✓ AfterBookingDelete (AFTER DELETE ON Booking)：当从 Booking 表成功删除一条预订记录后，自动将对应航班 (Flight 表) 的 booked_seats 减 1。

- **并发控制测试支持：**

  - ✓ Accounts 表：用于模拟并发场景下的账户余额操作。

- ✓ 通过设置不同的事务隔离级别 (Read Uncommitted, Read Committed, Repeatable Read, Serializable) 来验证和分析并发问题如脏读、不可重复读、丢失修改。

3. 系统界面设计

系统界面主要体现在 passengers.html 文件中，采用 Bootstrap 框架构建。

- **主页面布局：**
  - ✓ 顶部显示系统标题 "乘客管理系统" 及开发者信息。
  - ✓ 设有反馈消息区域，用于显示操作成功或失败的提示。
  - ✓ 右上角有 "添加新乘客" 按钮。
  - ✓ 主体部分为 "乘客列表" 表格。
- **乘客列表：**
  - ✓ 以表格形式展示乘客 ID、姓名、身份证号、电话、邮箱、常旅客号。
  - ✓ 每行末尾有 "编辑" 和 "删除" 操作按钮。
  - ✓ 数据加载时会显示加载动画和文字提示。
  - ✓ 若无数据，则提示 "没有找到乘客信息"。
- **乘客信息模态框 (passengerModal)：**
  - ✓ 用于添加新乘客和编辑现有乘客信息。
  - ✓ 模态框标题根据操作类型（添加/编辑）动态改变。
  - ✓ 包含表单字段：姓名、身份证号、联系电话、电子邮件（可选）、常旅客编号（可选）。
  - ✓ 表单字段均有标签和输入框，必填项有相应提示。
  - ✓ 身份证号和电话号码有格式校验。
  - ✓ 底部有 "关闭" 和 "保存" 按钮。

4. 系统物理模型

数据库物理模型定义了数据如何存储在数据库中，包括表、视图、索引等。

4.1 表 (Tables)

系统主要包含以下数据表（基于 DBLab3_22281188.sql 和 lab7_procedures_triggers.sql）：

- **Airline (航空公司)**
  - ✓ airline_id (INT, PK): 航空公司 ID
  - ✓ name (VARCHAR(100), NOT NULL): 名称
  - ✓ hq_location (VARCHAR(255)): 总部位置
  - ✓ contact_info (VARCHAR(50)): 联系方式
  - ✓ route_count (INT, DEFAULT 0): 航线数量
- **Airport (机场)**
  - ✓ airport_id (INT, PK): 机场 ID
  - ✓ airport_name (VARCHAR(100), NOT NULL): 机场名称
  - ✓ location (VARCHAR(255), NOT NULL): 机场位置
  - ✓ flight_count (INT, DEFAULT 0): 航班数量
- **Passenger (乘客)**
  - ✓ passenger_id (INT, PK, AUTO_INCREMENT): 乘客 ID
  - ✓ name (VARCHAR(100), NOT NULL): 姓名
  - ✓ id_card_number (VARCHAR(18), NOT NULL, UNIQUE): 身份证号
  - ✓ phone_number (VARCHAR(20), NOT NULL): 联系电话
  - ✓ email (VARCHAR(100), UNIQUE): 电子邮件
  - ✓ frequent_flyer_number (VARCHAR(50), UNIQUE): 常旅客编号
- **Route (航线)**
  - ✓ route_id (INT, PK): 航线 ID

- ✓ airline_id (INT, NOT NULL, FK -> Airline): 航空公司 ID
- ✓ origin (VARCHAR(100), NOT NULL): 起点
- ✓ destination (VARCHAR(100), NOT NULL): 终点
- ✓ duration (VARCHAR(20)): 飞行时长

- **Flight (航班)**
  - ✓ flight_id (INT, PK): 航班 ID
  - ✓ airline_id (INT, NOT NULL, FK -> Airline): 航空公司 ID
  - ✓ route_id (INT, NOT NULL, FK -> Route): 航线 ID
  - ✓ departure_time (DATETIME, NOT NULL): 起飞时间
  - ✓ arrival_time (DATETIME, NOT NULL): 抵达时间
  - ✓ departure_location (VARCHAR(100), NOT NULL): 起飞地点
  - ✓ destination_location (VARCHAR(100), NOT NULL): 目的地
  - ✓ total_seats (INT, NOT NULL): 座位总数
  - ✓ booked_seats (INT, DEFAULT 0): 已预订座位数
  - ✓ aircraft_model (VARCHAR(50)): 机型
- **Booking (预订)**
  - ✓ booking_id (INT, PK): 预订 ID
  - ✓ passenger_id (INT, NOT NULL, FK -> Passenger): 乘客 ID
  - ✓ flight_id (INT, NOT NULL, FK -> Flight): 航班 ID
  - ✓ seat_type (VARCHAR(20)): 座位类型
  - ✓ booking_date (DATETIME, NOT NULL): 预订日期
  - ✓ price (DECIMAL(10, 2), NOT NULL): 票价
  - ✓ payment_status (VARCHAR(20), NOT NULL, DEFAULT 'Pending'): 支付状态
- **Payment (支付)**
  - ✓ payment_id (INT, PK): 支付 ID
  - ✓ booking_id (INT, NOT NULL, FK -> Booking): 预订 ID
  - ✓ payment_method (VARCHAR(50), NOT NULL): 支付方式
  - ✓ payment_amount (DECIMAL(10, 2), NOT NULL): 支付金额

- ✓ payment_time (DATETIME, NOT NULL): 支付时间
- ✓ payment_status (VARCHAR(20), NOT NULL, DEFAULT 'Success'): 支付状态

- **Ticket (机票)**
  - ✓ ticket_id (INT, PK): 机票 ID
  - ✓ flight_id (INT, NOT NULL, FK -> Flight): 航班 ID
  - ✓ ticket_number (VARCHAR(50), NOT NULL, UNIQUE): 票号
  - ✓ seat_number (VARCHAR(10), NOT NULL): 座位号
  - ✓ passenger_id (INT, NOT NULL, FK -> Passenger): 乘客 ID
  - ✓ price (DECIMAL(10, 2), NOT NULL): 票价
- **FlightStatus (航班状态)**
  - ✓ flight_id (INT, PK, FK -> Flight): 航班 ID
  - ✓ status (VARCHAR(50), NOT NULL): 状态
  - ✓ update_time (DATETIME, NOT NULL): 更新时间
  - ✓ delay_reason (TEXT): 延误原因
  - ✓ cancellation_reason (TEXT): 取消原因
- **PassengerEmailAudit (乘客邮箱审计)**
  - ✓ audit_id (INT, PK, AUTO_INCREMENT): 审计 ID
  - ✓ passenger_id (INT): 乘客 ID
  - ✓ old_email (VARCHAR(100)): 旧邮箱
  - ✓ new_email (VARCHAR(100)): 新邮箱
  - ✓ change_timestamp (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): 变更时间
  - ✓ changed_by (VARCHAR(255), DEFAULT 'DB_TRIGGER'): 变更者
- **Accounts (账户 - 用于并发测试)**
  - ✓ account_id (INT, PK): 账户 ID
  - ✓ balance (DECIMAL(10, 2), NOT NULL): 余额

4.2 视图 (Views)

查询主要通过直接 SQL 语句或封装在存储过程中的 SQL 语句执行。

4.3 索引 (Indexes)

为提高查询性能和保证数据唯一性，系统在以下表的列上创建了索引：

- **Passenger**:
  - PRIMARY KEY (passenger_id)
  - UNIQUE KEY id_card_number (id_card_number)
  - UNIQUE KEY email (email)
  - UNIQUE KEY frequent_flyer_number (frequent_flyer_number)
  - INDEX idx_passenger_phone (phone_number)
  - INDEX idx_passenger_id_card (id_card_number) (通常 UNIQUE 约束会自动创建索引)
  - INDEX idx_passenger_email (email) (同上)
  - INDEX idx_passenger_ffn (frequent_flyer_number) (同上)
- **Route**:
  - PRIMARY KEY (route_id)
  - INDEX idx_route_airline (airline_id)
- **Flight**:
  - PRIMARY KEY (flight_id)
  - INDEX idx_flight_airline (airline_id)
  - INDEX idx_flight_route (route_id)
  - INDEX idx_flight_departure_time (departure_time)
- **Booking**:
  - PRIMARY KEY (booking_id)
  - INDEX idx_booking_passenger (passenger_id)
  - INDEX idx_booking_flight (flight_id)
- **Payment**:

- PRIMARY KEY (payment_id)
- INDEX idx_payment_booking (booking_id)

- **Ticket**:
  - PRIMARY KEY (ticket_id)
  - UNIQUE KEY ticket_number (ticket_number)
  - INDEX idx_ticket_flight (flight_id)
  - INDEX idx_ticket_passenger (passenger_id)
- **FlightStatus**:
  - PRIMARY KEY (flight_id)
- **PassengerEmailAudit**:
  - PRIMARY KEY (audit_id)
- **Accounts**:
  - PRIMARY KEY (account_id)

外键约束也通常会伴随相关列上索引的创建，以优化连接操作和约束检查。

5. 系统安全体系设计

5.1 用户管理与控制

其安全主要体现在以下方面：

- **数据库访问控制：**
  - ✓ 后端应用 (app.py) 通过配置文件 (DB_CONFIG) 中指定的数据库用户和密码连接到 MySQL 数据库。该用户的权限应被限制为仅执行应用所需的操作。
  - ✓ 并发测试脚本也使用类似的配置连接数据库。
- **数据校验：**
  - ✓ 前端通过 HTML5 属性和 JavaScript 进行输入校验（如必填项、身份证格式、邮箱格式）。

- ✓ 后端 API 接口对接收到的数据进行校验（如必填字段检查）。
- ✓ 数据库层面通过约束（NOT NULL, UNIQUE, FOREIGN KEY）保证数据完整性和一致性。

- **事务管理：**
  - ✓ 数据库操作（特别是写操作）被包含在事务中，确保操作的原子性。发生错误时会进行回滚。可以很好的处理并发事务。
  - ✓ 系统对 MySQL 的四种事务隔离级别进行了研究和测试，以理解和应对并发操作可能带来的数据不一致问题（脏读、不可重复读、丢失修改）。对数据一致性保护。

5.2 存储与恢复

- **数据存储：**所有业务数据持久化存储在 MySQL 数据库中。
- **数据恢复：**
  - ✓ MySQL 数据库管理系统自身提供了备份和恢复机制（如 mysqldump 工具、二进制日志等）。本系统的设计文档未详细说明自定义的备份恢复策略，通常依赖于数据库的标准功能。
  - ✓ 事务日志（如 InnoDB 的 redo log 和 undo log）确保了事务的持久性和原子性，能够在系统崩溃后进行恢复，将数据库恢复到一个一致的状态。
  - ✓ PassengerEmailAudit 表提供了一种针对特定数据（乘客邮箱变更）的审计和历史追溯能力，间接辅助了数据变更的跟踪。

6. 系统运行环境设计与部署结构

6.1 运行环境

- **操作系统**: macOS (开发和测试环境)
- **数据库管理系统**: MySQL 8.0
- **后端编程语言与框架**: Python 3.12, Flask
- **数据库连接库**: mysql-connector-python

- **API 测试工具**: Postman
- **并发测试环境**:
  - ✓ 多个 MySQL 命令行客户端实例
  - ✓ Python threading 模块编写的并发模拟脚本
- **文本编辑器/IDE**: Visual Studio Code

6.2 部署结构

系统采用典型的 B/S (Browser/Server) 架构：

1. **客户端 (Browser)**: 用户通过 Web 浏览器访问前端界面 (passengers.html)。前端通过 HTTP 请求与后端 Flask 应用进行交互。
2. **应用服务器 (Server)**:
   - 运行 Python Flask 应用 (app.py)。
   - 处理来自客户端的 HTTP 请求。
   - 执行业务逻辑。
   - 通过 mysql-connector-python 库连接并操作 MySQL 数据库。
   - 向客户端返回 JSON 数据或渲染 HTML 页面。
3. **数据库服务器 (Database Server)**:
   - 运行 MySQL 8.0 数据库服务。
   - 存储系统所有数据。
   - 执行 SQL 查询、存储过程和触发器。

7. 源代码列表及说明

系统主要由以下源代码文件和脚本构成：

- **app.py**:
  - ✓ **说明**: Python Flask 后端应用程序。定义了所有 API 端点，包括直接操作数据库的 API 和调用存储过程的 API。处理 HTTP 请求，连接数据库，执行业务逻辑，并返回 JSON 响应或 HTML 页面。

- **templates/passengers.html**:
  - ✓ **说明**: 系统的前端用户界面。使用 HTML、CSS（Bootstrap）和 JavaScript 构建。负责展示乘客列表，提供添加、编辑、删除乘客的表单和交互逻辑，并通过 AJAX 与后端 API 通信。
- **database_scripts/DBLab3_22281188.sql**:
  - ✓ **说明**: 数据库模式定义 SQL 脚本。包含创建所有数据表（Airline, Airport, Passenger, Route, Flight, Booking, Payment, Ticket, FlightStatus）及其约束（主键、外键、唯一约束、NOT NULL 等）和索引的语句。
- **database_scripts/DBLab3_Data_Insert.sql**:
  - ✓ **说明**: 初始数据插入 SQL 脚本。用于向已创建的表中填充示例数据，方便系统测试和演示。
- **database_scripts/lab7_procedures_triggers.sql**:
  - ✓ **说明**: 存储过程和触发器定义 SQL 脚本。包含创建实验所需的存储过程（GetPassengerAndBookings, AddPassengerViaSP, DeletePassengerViaSP, UpdatePassengerEmailViaSP）和触发器（AfterBookingInsert, AfterPassengerEmailUpdate, AfterBookingDelete），以及 PassengerEmailAudit 表的创建语句。
- **concurrency_tests/isolation_level_setup.sql**:
  - ✓ **说明**: 并发测试环境设置 SQL 脚本。用于创建 Accounts 表并插入初始数据，以及提供设置不同 MySQL 事务隔离级别的示例命令，供并发测试使用。
- **concurrency_tests/dirty_read_test.py**:
  - ✓ **说明**: Python 脚本，用于模拟和测试数据库在不同隔离级别下的“脏读”现象。使用 threading 模块创建并发事务。
- **concurrency_tests/non_repeatable_read_test.py**:
  - ✓ **说明**: Python 脚本，用于模拟和测试数据库在不同隔离级别下的“不可重复读”现象。
- **concurrency_tests/lost_update_test.py**:

- ✓ **说明**: Python 脚本，用于模拟和测试数据库在不同隔离级别下的“丢失修改”现象，并演示原子操作如何避免此问题。