用户安装与使用手册

一、环境准备与项目设置

1. **确认实验环境：**
   - **MySQL服务器**：已安装并正在运行。确保您有权限创建数据库、表、存储过程和触发器（通常使用`root`用户或具有相应权限的用户）。
   - **Python 3.x**：已安装。
   - **Flask 和 mysql-connector-python**：如果尚未安装，请通过pip安装：

     ```
     pip install Flask mysql-connector-python
     ```

   - **API测试工具**：如Postman或curl命令行。
   - **代码编辑器/IDE**：如VS Code。
2. **项目文件结构：**
   - 确保项目文件按照实验报告中建议的目录结构存放：

     ```
     AirlineApp/
     ├── database_scripts/
     |   ├── DBLab3_22281188.sql          # 数据库模式定义
     |   ├── DBLab3_Data_Insert.sql       # 初始数据插入
     |   └── lab7_procedures_triggers.sql # 存储过程和触发器
     ├── templates/
     |   └── passengers.html              # 前端页面
     ├── app.py                           # Python Flask应用
     ├── concurrency_tests/
     |   ├── isolation_level_setup.sql    # 设置隔离级别和测试账户表
     |   ├── dirty_read_test.py           # 脏读测试脚本
     |   ├── non_repeatable_read_test.py # 不可重复读测试脚本
     |   ├── lost_update_test.py          # 丢失更新测试脚本
     |   └── README_Lab7.md               # 并发测试说明
     ```

3. **配置数据库连接：**
   - 打开`app.py`文件，找到`DB_CONFIG`字典。
   - **重要**：将其中的`'password': 'Aa585891'`修改为您自己MySQL用户的正确密码。如果用户名或主机不是`root`和`localhost`，也请一并修改。

```
DB_CONFIG = {
    'user': 'root',
    'password': 'YOUR_MYSQL_PASSWORD', # <--- 修改这里
    'host': 'localhost',
    'database': 'AirlineDB',
    'raise_on_warnings': True
}
```

- 同样，检查 `concurrency_tests/` 目录下的三个Python脚本（`dirty_read_test.py`，`non_repeatable_read_test.py`，`lost_update_test.py`），确保它们顶部的 `DB_CONFIG` 也已更新为正确的密码。

二、数据库初始化

您需要按顺序执行SQL脚本来创建数据库模式、插入初始数据，然后创建存储过程和触发器。

1. **登录MySQL：**
   - 打开您的MySQL客户端（如命令行、MySQL Workbench、DBeaver等）。
   - 使用具有足够权限的用户登录（例如 `root`）。
2. **创建数据库（如果尚不存在）：**
   - 在 `DBLab3_22281188.sql` 脚本中，有一行被注释掉的创建数据库的语句。如果您的 `AirlineDB` 数据库还不存在，可以取消注释并执行，或者手动创建：

     ```
     CREATE DATABASE IF NOT EXISTS AirlineDB CHARACTER SET utf8mb4
     COLLATE utf8mb4_unicode_ci;
     USE AirlineDB;
     ```

   - 如果数据库已存在，请确保您正在使用它：

     ```
     USE AirlineDB;
     ```

3. **执行SQL脚本：**
   - **顺序至关重要！**
   - **第一步：创建表结构**

    - 在MySQL客户端中，打开并执行 `database_scripts/DBLab3_22281188.sql` 文件的全部内容。这将创建所有必要的表和它们之间的关系。
  - **第二步：插入初始数据**
    - 接着，打开并执行 `database_scripts/DBLab3_Data_Insert.sql` 文件的全部内容。这将为您的表填充一些初始记录。
  - **第三步：创建存储过程和触发器**
    - 最后，打开并执行 `database_scripts/lab7_procedures_triggers.sql` 文件的全部内容。这将创建实验七所需的存储过程和触发器。
  - **检查：** 执行完每个脚本后，检查是否有错误信息。您可以执行一些简单的 `SELECT` 语句来确认表已创建并且数据已插入，例如：

```
SELECT * FROM Passenger LIMIT 5;
SELECT * FROM Flight LIMIT 5;
SHOW PROCEDURE STATUS WHERE Db = 'AirlineDB'; -- 查看已创建的存储过程
SHOW TRIGGERS FROM AirlineDB; -- 查看已创建的触发器
```

三、启动后端Flask应用

1. **打开终端或命令行：**
   - 导航到您的项目根目录 `AirlineApp_Lab7/`。
2. **运行Flask应用：**
   - 执行以下命令：

```
python app.py
```

   - 如果一切正常，您应该会看到类似以下的输出，表明Flask开发服务器正在运行（通常在 `http://127.0.0.1:5000/`）：

```
 * Serving Flask app 'app'
 * Debug mode: on
 * Running on [http://127.0.0.1:5000](http://127.0.0.1:5000)
(Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: xxx-xxx-xxx
```

3. 而后可以开始使用`航空管理系统`进行数据的插入和删除等一系列操作

四、(可选)测试存储过程和触发器

如果需要测试，可以follow下面的操作：

A. 通过MySQL客户端手动测试

您可以直接在MySQL客户端中调用存储过程并观察触发器的行为，如`lab7_procedures_triggers.sql`文件末尾的注释示例所示。

1. **测试存储过程 `GetPassengerAndBookings`：**

   ```
   USE AirlineDB;
   CALL GetPassengerAndBookings(1001); -- 假设乘客ID 1001 存在
   ```

   观察返回的乘客及其预订信息。

2. **测试存储过程 `AddPassengerViaSP` 和触发器 `AfterPassengerEmailUpdate`（如果添加时提供了邮箱）：**

   ```
   SET @new_id = 0;
   CALL AddPassengerViaSP('SP测试用户甲', '987654321012345678X',
   '13011112222', 'sptest@example.com', 'SPFL007', @new_id);
   SELECT @new_id; -- 查看新生成的乘客ID
   SELECT * FROM Passenger WHERE passenger_id = @new_id;
   -- 如果添加时提供了邮箱，并且之后用UpdatePassengerEmailViaSP更新了邮箱，可以检查PassengerEmailAudit表
   ```

3. **测试触发器 `AfterBookingInsert`：**

```
-- 1. 查看航班3001的初始booked_seats
SELECT flight_id, booked_seats FROM Flight WHERE flight_id =
3001;
-- 2. 插入一条新的预订 (确保booking_id唯一, 例如5001)
INSERT INTO Booking (booking_id, passenger_id, flight_id,
seat_type, booking_date, price, payment_status)
VALUES (5001, 1002, 3001, 'Economy', NOW(), 750.00, 'Paid');
-- 3. 再次查看booked_seats, 应已增加1
SELECT flight_id, booked_seats FROM Flight WHERE flight_id =
3001;
```

4. 测试存储过程 **`UpdatePassengerEmailViaSP`** 和触发器 **`AfterPassengerEmailUpdate`**:

```
-- 1. 查看乘客1001的当前邮箱和PassengerEmailAudit表关于此乘客的记录
SELECT email FROM Passenger WHERE passenger_id = 1001;
SELECT * FROM PassengerEmailAudit WHERE passenger_id = 1001 ORDER
BY change_timestamp DESC;
-- 2. 调用存储过程更新邮箱
CALL UpdatePassengerEmailViaSP(1001,
'new.sp.email.updated@example.com');
-- 3. 再次查看邮箱和审计表, 应有新记录
SELECT email FROM Passenger WHERE passenger_id = 1001;
SELECT * FROM PassengerEmailAudit WHERE passenger_id = 1001 ORDER
BY change_timestamp DESC;
```

5. 测试触发器 **`AfterBookingDelete`**:

```
-- (接步骤3, 此时航班3001的booked_seats应为初始值+1)
-- 1. 删除之前插入的预订记录 (booking_id = 5001)
DELETE FROM Booking WHERE booking_id = 5001;
-- 2. 再次查看booked_seats, 应已减少1, 恢复到初始值
SELECT flight_id, booked_seats FROM Flight WHERE flight_id =
3001;
```

6. 测试存储过程 **`DeletePassengerViaSP`**:

```
-- 使用之前通过SP添加的乘客ID（例如@new_id的值）
-- CALL DeletePassengerViaSP(@new_id);
-- SELECT * FROM Passenger WHERE passenger_id = @new_id; -- 应返回空结果
-- 注意：如果该乘客有关联的预订或机票，且外键设置为RESTRICT，则删除会失败。
-- SP本身未处理此错误，错误会由数据库抛出。
```

B. 通过API端点测试（主要测试方式）

使用Postman或curl测试`app.py`中定义的与存储过程相关的API端点。确保Flask应用正在运行。

- **查询乘客详情（调用`GetPassengerAndBookings`）**
  - **Method:** `GET`
  - **URL:** `http://127.0.0.1:5000/api/sp/passengers/1001/details`（将1001替换为存在的乘客ID）
  - **预期：** 返回包含乘客信息及其预订列表的JSON。
- **添加新乘客（调用`AddPassengerViaSP`）**
  - **Method:** `POST`
  - **URL:** `http://127.0.0.1:5000/api/sp/passengers`
  - **Headers:** `Content-Type: application/json`
  - **Body (raw JSON):**

    ```
    {
        "name": "API SP测试用户",
        "id_card_number": "102030405060708O9X",
        "phone_number": "13900001111",
        "email": "api.sp@example.com",
        "frequent_flyer_number": "APISPFLY01"
    }
    ```

  - **预期：** 返回成功消息和新乘客的`passenger_id`（HTTP 201）。同时，可以去数据库检查`Passenger`表是否添加了此记录。
- **更新乘客邮箱（调用`UpdatePassengerEmailViaSP`）**
  - **Method:** `PUT`

- URL: `http://127.0.0.1:5000/api/sp/passengers/1001/email` (将1001替换为存在的乘客ID)
- Headers: `Content-Type: application/json`
- Body (raw JSON):

```
{
    "email": "updated.api.sp@example.com"
}
```

- **预期：** 返回成功消息。去数据库检查`Passenger`表对应乘客的邮箱是否更新，并检查`PassengerEmailAudit`表是否有新的审计记录。

- **删除乘客 (调用 `DeletePassengerViaSP`)**
  - Method: `DELETE`
  - URL: `http://127.0.0.1:5000/api/sp/passengers/<passenger_id_to_delete>` (替换为要删除的乘客ID，例如上一步通过API添加的乘客ID)
  - **预期：** 返回成功消息。去数据库检查`Passenger`表是否已删除该记录。

触发器测试的API验证：

- **`AfterBookingInsert` / `AfterBookingDelete`**：
  1. 通过API `GET /api/flights/<flight_id>` (如果Lab5有此API) 或直接数据库查询，获取某航班（如3001）的`booked_seats`。
  2. 通过API `POST /api/bookings` (如果Lab5有此API，或手动构造一个调用普通插入的API) 添加一个指向该航班的预订。
  3. 再次获取该航班的`booked_seats`，验证是否增加。
  4. 通过API `DELETE /api/bookings/<booking_id>` 删除该预订。
  5. 再次获取`booked_seats`，验证是否减少。
     - 如果Lab5没有直接操作Booking的API，可以通过操作Passenger的SP间接触发，或者如A部分在MySQL客户端直接操作Booking表来验证，然后通过API查询Flight信息。
- **`AfterPassengerEmailUpdate`**：已在上面“更新乘客邮箱”的API测试中通过检查`PassengerEmailAudit`表间接验证。

五、（可选）测试并发控制

这部分主要通过运行`concurrency_tests/`目录下的Python脚本来模拟和观察。

1. **准备测试表和数据：**
    - 在MySQL客户端中，执行`concurrency_tests/isolation_level_setup.sql`脚本。这将创建`Accounts`表并插入一些初始数据，供并发测试使用。

      ```
      USE AirlineDB;
      -- (执行 isolation_level_setup.sql 的内容)
      ```

2. **运行并发测试脚本：**
    - 打开新的终端窗口（不要关闭Flask应用的终端）。
    - 导航到`AirlineApp_Lab7/concurrency_tests/`目录。
    - **重要：** 再次确认这些脚本中的`DB_CONFIG`密码已正确设置。
    - **逐个运行脚本：**
        - **脏读测试：**

          ```
          python dirty_read_test.py
          ```

          观察脚本输出。脚本内部会分别测试在事务B为`READ UNCOMMITTED`和`READ COMMITTED`隔离级别下的情况。留意线程B读取到的余额是否为线程A未提交的修改。
        - **不可重复读测试：**

          ```
          python non_repeatable_read_test.py
          ```

          观察脚本输出。脚本内部会分别测试在事务A为`READ COMMITTED`和`REPEATABLE READ`隔离级别下的情况。留意线程A两次读取余额是否一致。
        - **丢失修改测试：**

          ```
          python lost_update_test.py
          ```

观察脚本输出。脚本会模拟非原子的“读-改-写”操作，并对比在`READ COMMITTED`和`REPEATABLE READ`隔离级别下的结果，以及使用原子`UPDATE`操作的结果。留意最终`booked_seats`是否正确增加了2（如果两个线程都成功）。

3. **手动并发测试（可选，加深理解）：**
    - 按照`concurrency_tests/README_Lab7.md`中的指导，打开多个MySQL客户端窗口。
    - 在每个窗口中手动设置不同的事务隔离级别（`SET SESSION TRANSACTION ISOLATION LEVEL ...;`）。
    - 手动执行SQL语句，模拟`README_Lab7.md`中描述的脏读、不可重复读、幻读（如果想尝试）的步骤，观察现象。

六、（可选）前端页面集成与测试

如果您想从`passengers.html`前端页面调用新的基于存储过程的API：

1. **修改`passengers.html`中的JavaScript：**
    - 找到相关的API调用函数，例如`savePassenger`、`deletePassenger`。
    - 您可以选择修改现有的`apiUrl`指向新的SP端点（例如，将`/api/passengers`改为`/api/sp/passengers`），或者创建新的函数来调用SP端点。
    - 确保请求的数据格式和处理响应的逻辑与新的SP API端点兼容。例如，`GET /api/sp/passengers/<id>/details`返回的数据结构可能与原来的`GET /api/passengers/<id>`不同。
2. **在浏览器中测试：**
    - 刷新`http://127.0.0.1:5000/`页面。
    - 通过页面上的“添加”、“编辑”、“删除”按钮进行操作，观察是否成功调用了新的后端API，以及数据是否按预期变化。同时可以查看Flask应用的控制台输出来确认请求路径。