哈尔滨工业大学

**<<数据库系统>>**

**实验报告一**

**(2024年度秋季学期)**

|  |  |
| --- | --- |
| **姓名：** |  |
| **学号：** |  |
| **学院：** | **计算学部** |
| **教师：** | **张浩** |

实验一

## 实验目的

1. 理解数据库设计原理：通过实际设计和创建数据库，掌握关系数据库的基本概念，包括实体、属性、主键、外键、关系等。

2. 掌握数据库操作：学习如何在 MySQL 中进行数据库的创建、表的设计与优化，包括视图、索引的使用。

3. 实现常见的数据库操作：熟悉 SQL 语句的使用，包括插入、查询、更新、删除等基本操作，以及如何进行连接查询、嵌套查询、分组查询等复杂操作。

4. 数据完整性和约束管理：通过实验，理解如何在数据库中定义和维护数据的完整性，设计主键、外键约束，确保数据的准确性和一致性。

5. 提高数据库性能：通过使用视图和索引，优化数据库查询速度，提高系统的响应效率。

## 二、实验环境

1. 硬件环境：使用 Windows 操作系统的 x86 架构计算机。

2. 软件环境：

- 操作系统：Windows 11

- 数据库系统：MySQL Server 8.0

- 数据库管理工具：MySQL Workbench 8.0，用于设计、管理数据库，以及编写和执行 SQL 查询。

- 开发工具：IntelliJ IDEA，用于编写和管理 Java 程序代码。

- Java 环境：Java SE 8 或更高版本，用于开发和运行 Java 应用程序。

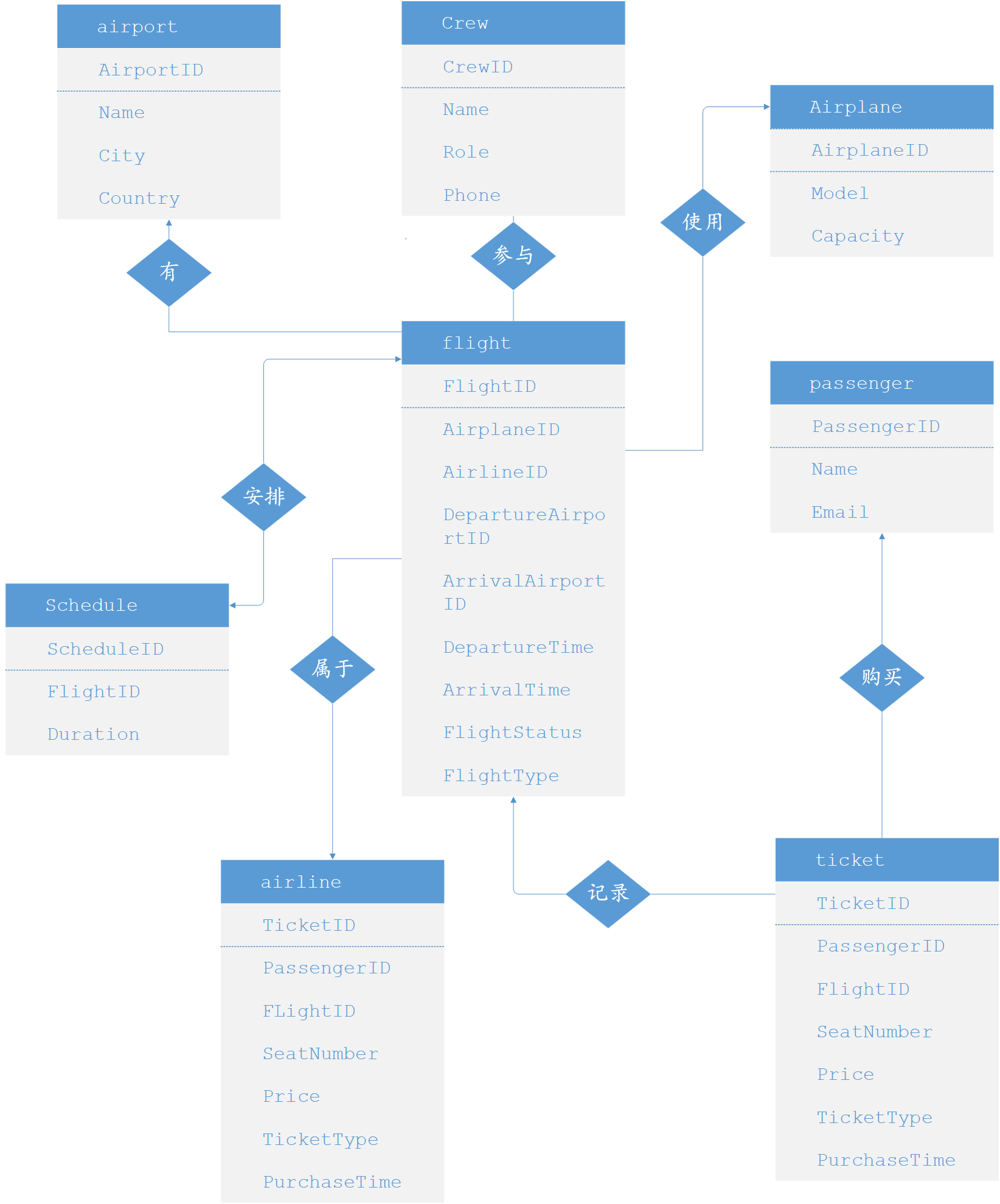
3. 网络配置：

- 本地数据库连接：通过 MySQL Workbench 配置和管理本地数据库。

## 三、实验过程及结果

**3.1 数据库的设计与描述**

根据本实验的要求，设计数据库的E-R图如下：



在 E-R 图中，以下是主要实体的定义及其主键、属性等信息：

1. Airplane（飞机）

- 主键：AirplaneID（INT，AUTO\_INCREMENT）

- 属性：

- Model（VARCHAR(100)）：飞机型号

- Capacity（INT）：座位容量

2. Airline（航空公司）

- 主键：AirlineID（INT，AUTO\_INCREMENT）

- 属性：

- Name（VARCHAR(100)）：航空公司名称

- Headquarters（VARCHAR(100)）：总部地址

3. Airport（机场）

- 主键：AirportID（INT，AUTO\_INCREMENT）

- 属性：

- Name（VARCHAR(100)）：机场名称

- City（VARCHAR(100)）：所在城市

- Country（VARCHAR(100)）：所在国家

4. Passenger（乘客）

- 主键：PassengerID（INT，AUTO\_INCREMENT）

- 属性：

- Name（VARCHAR(100)）：乘客姓名

- Phone（VARCHAR(15)）：联系电话

- Email（VARCHAR(100)）：电子邮件

5. Crew（机组成员）

- 主键：CrewID（INT，AUTO\_INCREMENT）

- 属性：

- Name（VARCHAR(100)）：机组成员姓名

- Role（VARCHAR(50)）：成员角色（如机长、空乘）

- Phone（VARCHAR(15)）：联系电话

6. Flight（航班）

- 主键：FlightID（VARCHAR(10)）

- 属性：

- AirplaneID（INT）：关联的飞机 ID（外键，引用 Airplane 表）

- AirlineID（INT）：关联的航空公司 ID（外键，引用 Airline 表）

- DepartureAirportID（INT）：起飞机场 ID（外键，引用 Airport 表）

- ArrivalAirportID（INT）：到达机场 ID（外键，引用 Airport 表）

- DepartureTime（DATETIME）：起飞时间

- ArrivalTime（DATETIME）：到达时间

- FlightStatus（VARCHAR(50)）：航班状态（如计划、延误、取消）

- FlightType（VARCHAR(20)）：航班类型（如国内、国际）

7. Ticket（票务）

- 主键：TicketID（VARCHAR(10)）

- 属性：

- PassengerID（INT）：关联的乘客 ID（外键，引用 Passenger 表）

- FlightID（VARCHAR(10)）：关联的航班 ID（外键，引用 Flight 表）

- SeatNumber（VARCHAR(10)）：座位号

- Price（DECIMAL(10, 2)）：票价

- TicketType（VARCHAR(20)）：票务类型（如经济舱、商务舱）

- PurchaseTime（DATETIME）：购票时间

8. Schedule（航班安排）

- 主键：ScheduleID（INT，AUTO\_INCREMENT）

- 属性：

- FlightID（VARCHAR(10)）：关联的航班 ID（外键，引用 Flight 表）

- Duration（DECIMAL(4, 2)）：飞行时长（单位：小时）

关系描述：

1. Flight 和 Schedule （一对一关系）

- 描述：每个航班有唯一的航班安排。

- 外键：Schedule 表中的 FlightID 作为外键引用 Flight 表的 FlightID。

- 示例：一个航班 CR345 对应一个航班安排，记录了该航班的飞行时长。

2. Airline 和 Flight （一对多关系）

- 描述：一个航空公司拥有多个航班。

- 外键：Flight 表中的 AirlineID 作为外键引用 Airline 表的 AirlineID。

- 示例：某个航空公司 AirlineA 拥有多个航班，如 CR345、AB567。

3. Airplane 和 Flight （多对一关系）

- 描述：每个航班使用一架特定的飞机。

- 外键：Flight 表中的 AirplaneID 作为外键引用 Airplane 表的 AirplaneID。

4. Flight 和 Airport （多对一关系）

- 描述：每个航班有一个起飞机场和一个到达机场。

- 外键：Flight 表中的 DepartureAirportID 和 ArrivalAirportID 分别作为外键引用 Airport 表的 AirportID。

5. Ticket 和 Passenger （多对一关系）

- 描述：一个乘客可以购买多张票。

- 外键：Ticket 表中的 PassengerID 作为外键引用 Passenger 表的 PassengerID。

6. Ticket 和 Flight （多对一关系）

- 描述：一个航班可以有多个票务记录。

- 外键：Ticket 表中的 FlightID 作为外键引用 Flight 表的 FlightID。

- 示例：航班 CR345 有多张票务记录，代表多个乘客的预订信息。

7. Flight 和 Crew（多对多关系）

- 描述：一个航班可以有多个机组成员，每个机组成员可以参与多个航班。

- 示例：航班 CR345 有多名机组成员，如 Captain Smith 和 Attendant Lee。

**3.2 检查数据库创建与规范化要求**

3.2.1范式检查：

① 第一范式 (1NF)：每个表的字段都保持原子性，没有复合属性。所有字段都是不可进一步分解的。

② 第二范式 (2NF)：所有非主键字段都完全依赖于主键。每个表的外键和主键关系均设计合理，没有部分依赖。例如：Ticket 表的非主键字段（SeatNumber、Price 等）都完全依赖于 TicketID 主键。

③ 第三范式 (3NF)：没有传递依赖。表中的非主键字段不依赖其他非主键字段。例如：Flight 表中的 DepartureTime 和 ArrivalTime 都依赖于主键 FlightID，不存在依赖其他非主键字段的情况。

3.2.2约束和完整性检查

① 主键约束：每个表都定义了主键来确保记录的唯一性（如 FlightID、TicketID、PassengerID 等）。

② 外键约束：外键完整性在各表之间通过 FOREIGN KEY 约束来实现，确保外键的引用一致性。例如：FlightID 在 Ticket 表中引用 Flight 表。PassengerID 在 Ticket 表中引用 Passenger 表。

③ 空值约束：重要字段如 FlightID、AirplaneID、DepartureTime、SeatNumber 等都被设置为 NOT NULL，确保关键数据不会为空。

**3.3 创建视图与索引**

- 步骤：

- 创建了常用查询的视图，以简化复杂的查询操作：

- FlightCrewPassengerView：查询航班的详细信息，包括机组成员和乘客。

CREATE VIEW FlightCrewPassengerView AS

SELECT f.FlightID, f.DepartureTime, f.ArrivalTime,

c.Name AS CrewName, c.Role AS CrewRole,

p.Name AS PassengerName, t.SeatNumber

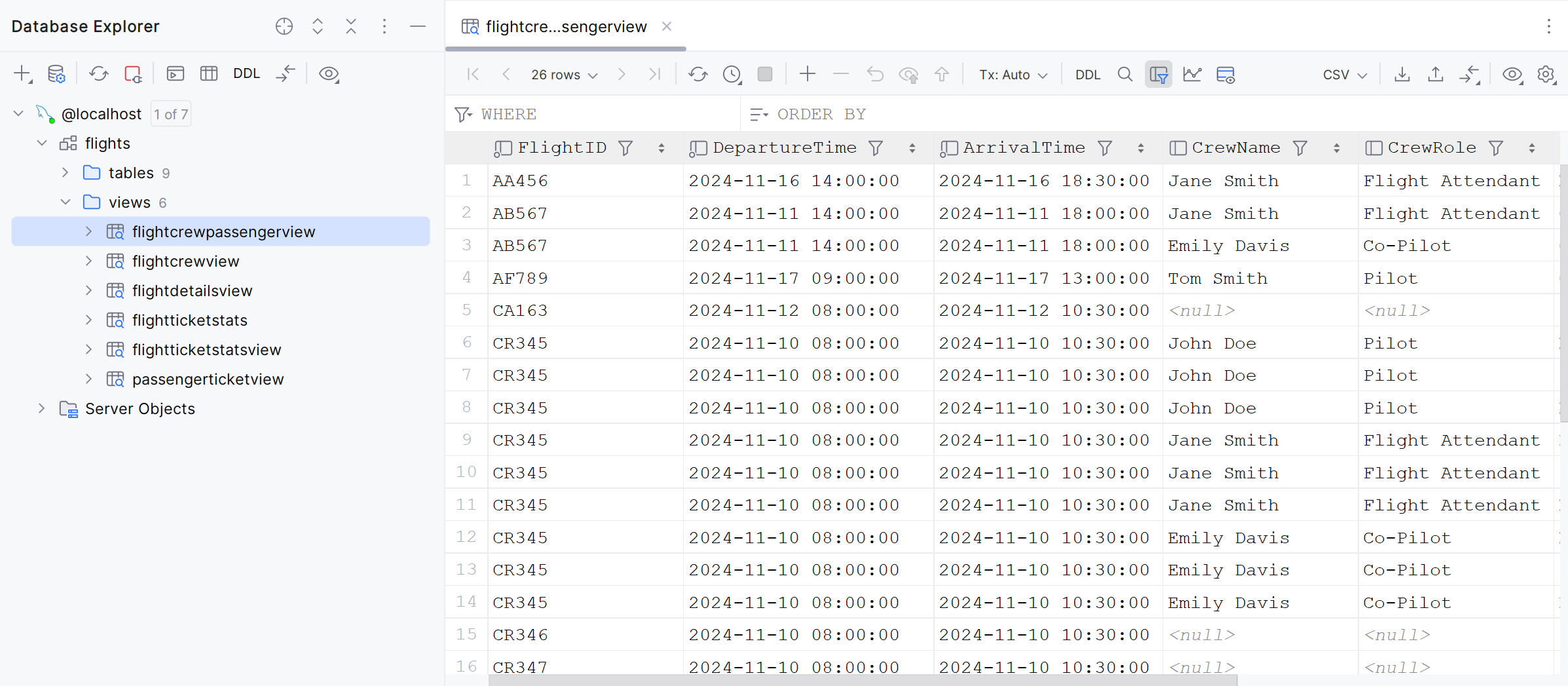
FROM Flight f

LEFT JOIN Flight\_Crew fc ON f.FlightID = fc.FlightID

LEFT JOIN Crew c ON fc.CrewID = c.CrewID

LEFT JOIN Ticket t ON f.FlightID = t.FlightID

LEFT JOIN Passenger p ON t.PassengerID = p.PassengerID;



- FlightTicketStatsView：统计每个航班的乘客数量和总票务收入。

CREATE VIEW FlightTicketStatsView AS

SELECT f.FlightID, COUNT(t.TicketID) AS PassengerCount, SUM(t.Price) AS TotalRevenue, AVG(t.Price) AS AveragePrice

FROM Flight f

LEFT JOIN Ticket t ON f.FlightID = t.FlightID

GROUP BY f.FlightID;

图形用户界面

中度可信度描述已自动生成

- 为常用的查询属性创建索引，以提高查询效率：

- 为 Flight 表的 DepartureTime 和 ArrivalTime 创建索引，方便根据时间筛选航班。

CREATE INDEX idx\_flight\_departuretime ON Flight(DepartureTime);

CREATE INDEX idx\_flight\_arrivaltime ON Flight(ArrivalTime);

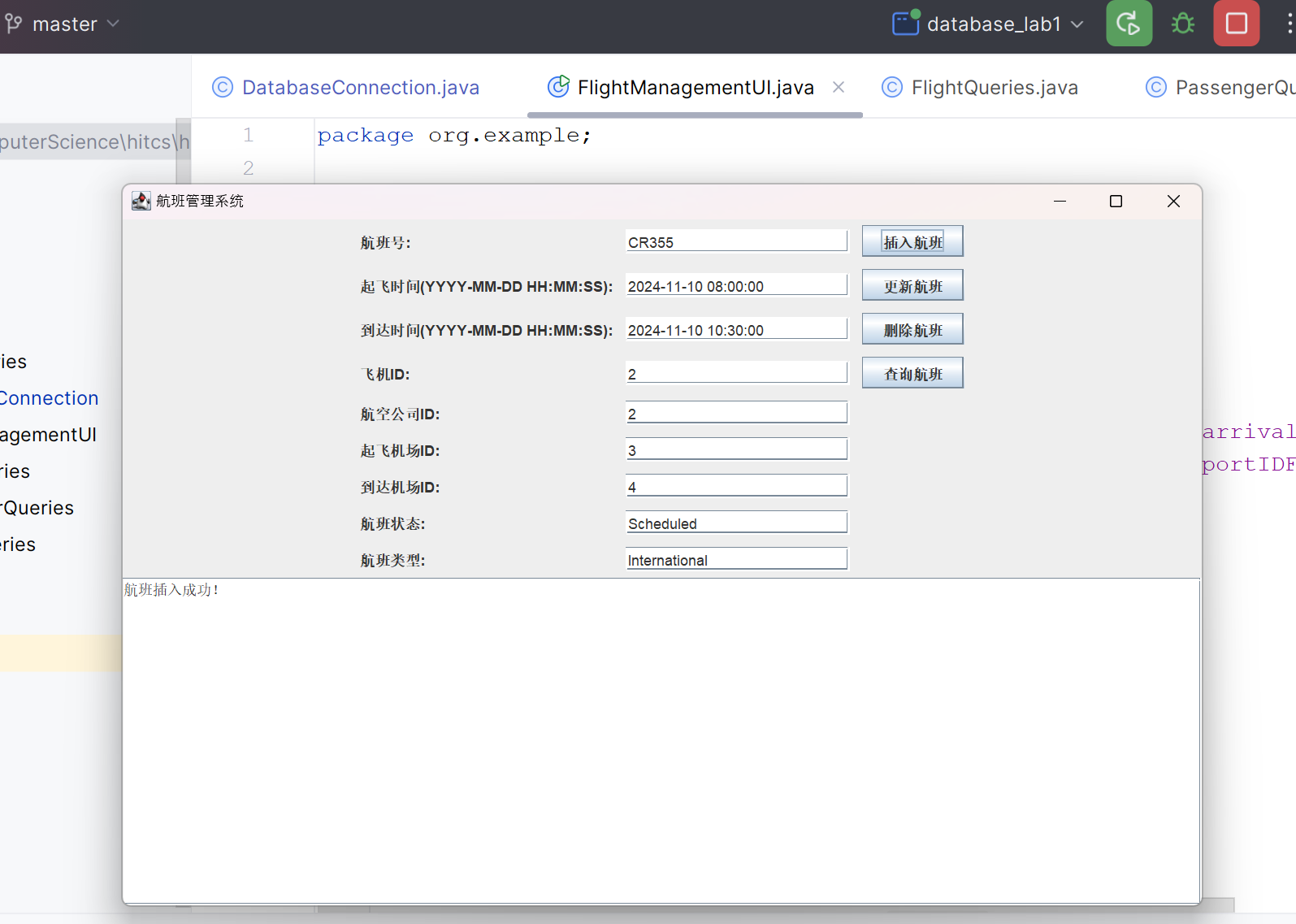
- 为 Ticket 表的 Price 创建索引，优化价格查询。

CREATE INDEX idx\_ticket\_price ON Ticket(Price);

**3.4 数据的插入与删除操作**

- 步骤：使用 SQL 语句进行数据的插入、更新和删除操作，确保遵循数据库完整性约束。

正常插入完整的记录：



- 插入数据时，若主键重复或插入空值，则系统提示错误信息并拒绝插入。

图形用户界面

描述已自动生成

图形用户界面

描述已自动生成

正常删除情况：

图形用户界面

描述已自动生成

- 删除航班记录时，若该航班存在票务或机组记录关联，则拒绝删除，提示外键约束错误。

图形用户界面, 应用程序

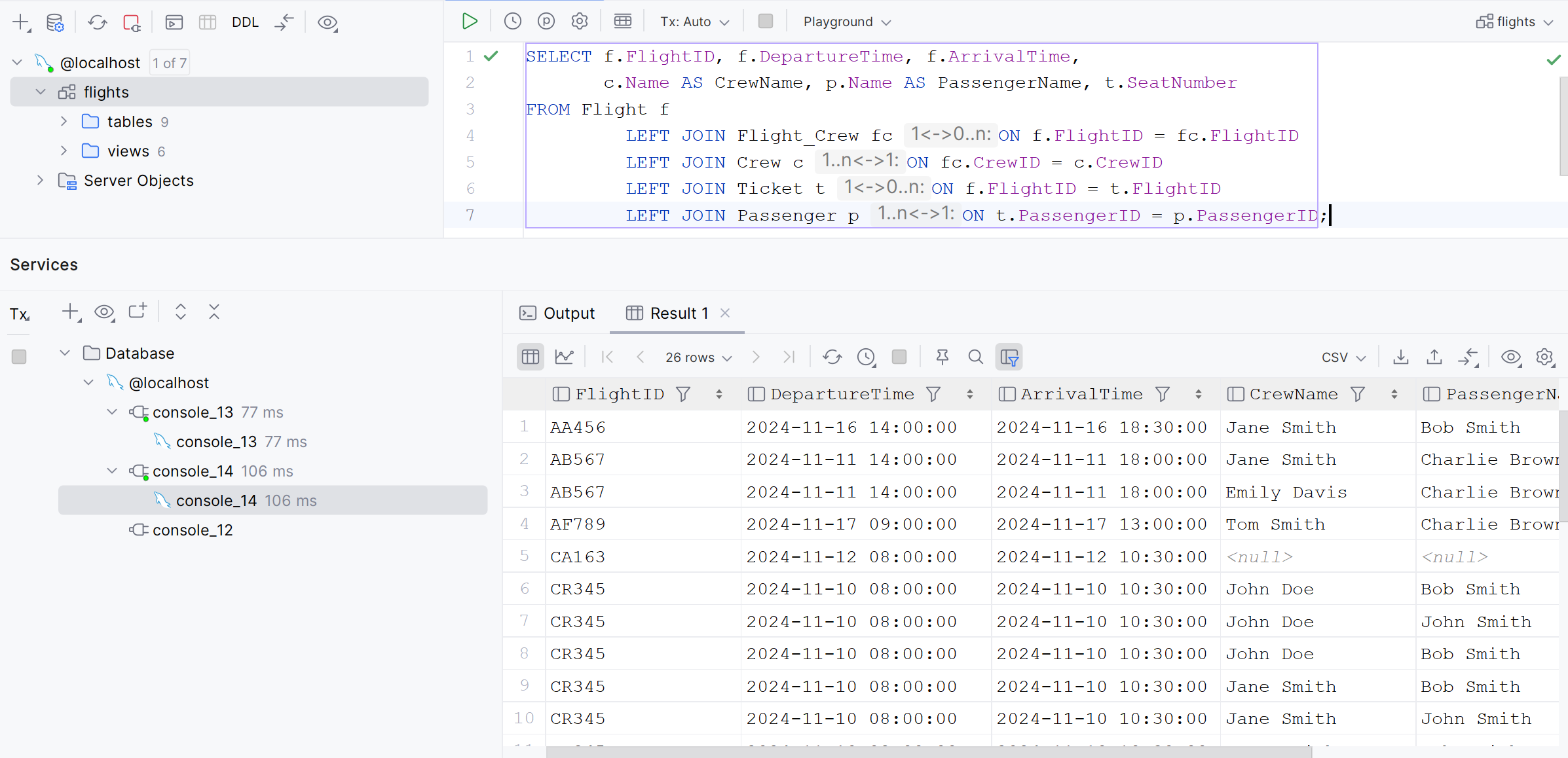
描述已自动生成

**3.5 复杂查询的实现**

- 步骤：使用 SQL 语句实现了连接查询、嵌套查询、分组查询等功能，具体如下：

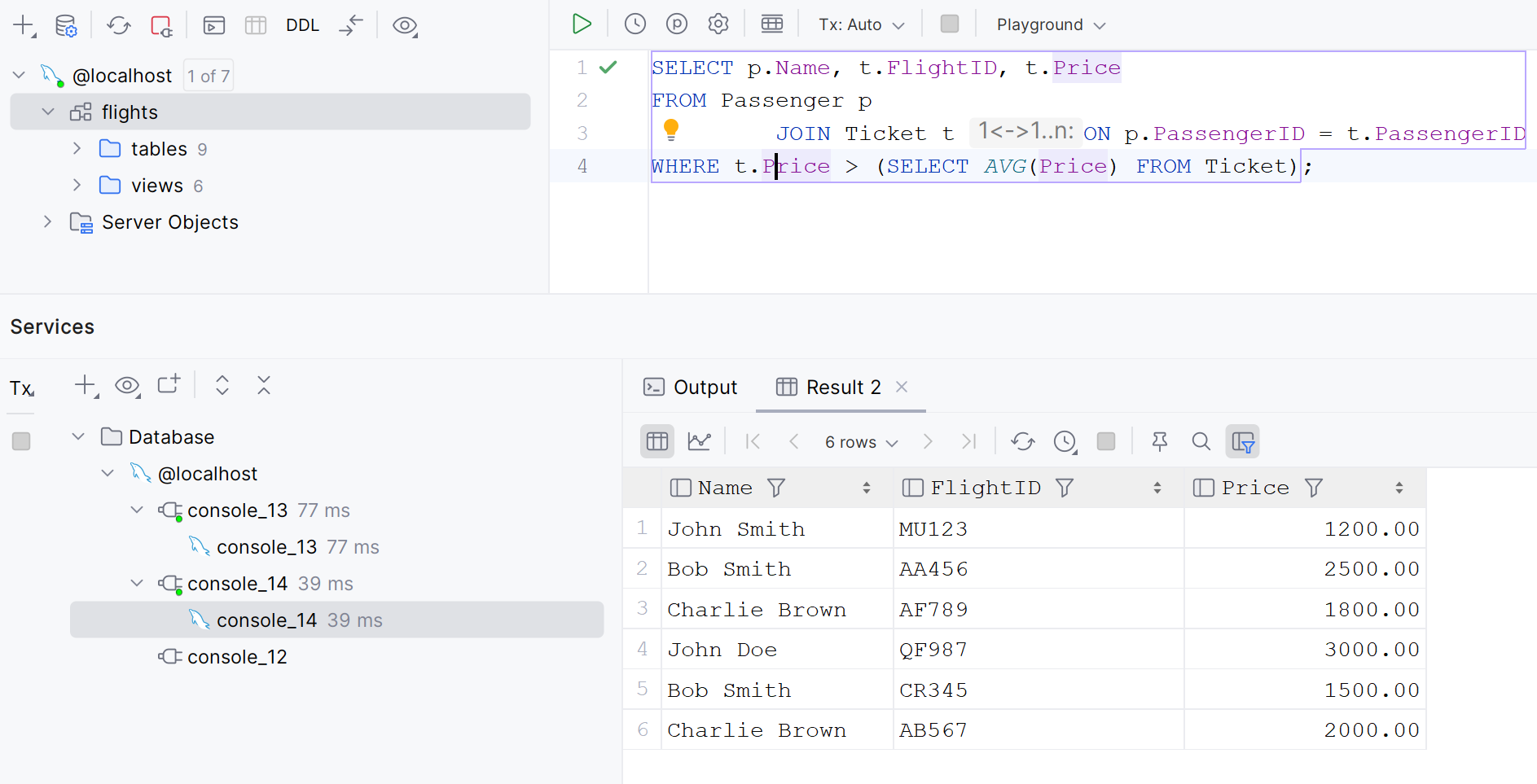
- 连接查询：通过连接 Flight、Ticket 和 Passenger 表，实现航班与乘客信息的关联查询，展示某一航班的所有乘客信息。

SELECT f.FlightID, f.DepartureTime, f.ArrivalTime,  
 c.Name AS CrewName, p.Name AS PassengerName, t.SeatNumber  
FROM Flight f  
 LEFT JOIN Flight\_Crew fc ON f.FlightID = fc.FlightID  
 LEFT JOIN Crew c ON fc.CrewID = c.CrewID  
 LEFT JOIN Ticket t ON f.FlightID = t.FlightID  
 LEFT JOIN Passenger p ON t.PassengerID = p.PassengerID;



- 嵌套查询：查询票价高于平均票价的乘客，使用子查询来计算平均票价，并筛选符合条件的记录。

SELECT p.Name, t.FlightID, t.Price  
FROM Passenger p  
 JOIN Ticket t ON p.PassengerID = t.PassengerID  
WHERE t.Price > (SELECT AVG(Price) FROM Ticket);



- 分组查询：统计每个航班的乘客数量和总收入，并使用 HAVING 子句筛选出乘客数量超过 2 的航班。

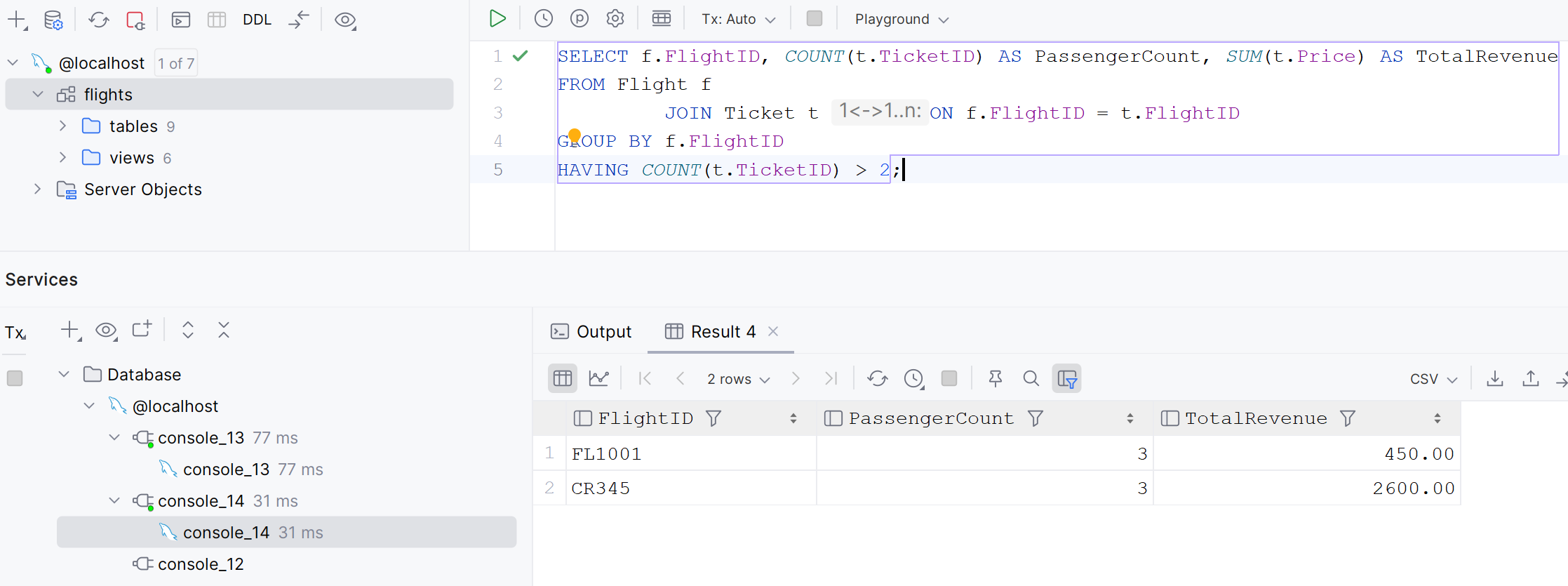
SELECT f.FlightID, COUNT(t.TicketID) AS PassengerCount, SUM(t.Price) AS TotalRevenue

FROM Flight f

JOIN Ticket t ON f.FlightID = t.FlightID

GROUP BY f.FlightID

HAVING COUNT(t.TicketID) > 2;



- 结果：所有查询均按要求完成，结果符合预期。分组查询帮助分析了航班的乘客人数和收入，查询效率在索引的支持下得到了优化。

**3.6 实验结果总结**

- 数据库结构符合设计要求，各个实体和关系清晰明了。

- 视图与索引的使用提升了查询效率，便于数据分析。

- 图形化用户界面使操作更加简便，改善了用户体验。

## 四、实验心得

问题 1：插入数据时出现数据类型不匹配

- 问题描述：在插入数据时，出现了数据类型不匹配的错误。例如，在插入航班时，AirlineID 应该为整数，但用户输入了字符串导致出错。

- 解决方法：在 Java 程序中增加了输入验证的逻辑，确保用户输入的 ID 和日期格式正确，并在遇到错误时提示用户重新输入。同时在前端界面中明确标注数据格式要求。

问题 2：视图和索引的创建和使用

- 问题描述：在创建视图时，由于没有完全理解视图的作用，一开始误以为视图可以替代所有查询需求。而在使用索引时，也遇到了索引创建后查询速度没有明显提升的问题。

- 解决方法：通过查阅资料，理解视图的主要作用是简化复杂查询，便于数据展示。视图不是物理存储数据，而是基于查询的逻辑结构。索引则用于提升查询速度，但需要根据查询条件的不同，选择性地创建索引。实验中重新调整了索引的使用场景，提升了查询效率。

问题 3：MySQL 用户权限配置的错误

- 问题描述：在配置数据库用户权限时，执行 GRANT 命令时遇到语法错误，导致权限无法正常授予。

- 解决方法：仔细阅读 MySQL 官方文档，确认了正确的 GRANT 语法，确保不再在 IDENTIFIED BY 中重复设置密码。之后顺利授予权限，并成功测试了数据库的访问和修改。