

Bilkent University

Department of Computer Engineering

CS 353 - Database Management Systems



Airline Company Data Management System

**Design Report**

Group 16

|  |  |
| --- | --- |
| Mert Aytöre | 21400923 |
| Oğuz Demir | 21201712 |
| Ayşe Berceste Dinçer | 21300957 |
| Mehmet Furkan Şahin | 21201385 |

04.04.2016

# Table of Contents

[5. Table of Contents 2](#_Toc447445836)

[1. REVISED E/R MODEL 5](#_Toc447445837)

[1.1. Changes Made in the E/R Diagram 5](#_Toc447445838)

[1.2. Revised E/R Diagram 7](#_Toc447445839)

[2. RELATION SCHEMAS 8](#_Toc447445840)

[2.1. Staff 8](#_Toc447445841)

[2.2. Staff Phones 8](#_Toc447445842)

[2.3. Reservation Authoritative 9](#_Toc447445843)

[2.4. Permissions 9](#_Toc447445844)

[2.5. Manager 10](#_Toc447445845)

[2.6. Salesperson 10](#_Toc447445846)

[2.7. Crew 11](#_Toc447445847)

[2.8. Pilot 11](#_Toc447445848)

[2.9. Flight Attendance 12](#_Toc447445849)

[2.10. City 12](#_Toc447445850)

[2.11. Airport 13](#_Toc447445851)

[2.12. Route 14](#_Toc447445852)

[2.13. Plane Type 14](#_Toc447445853)

[2.14. Plane 15](#_Toc447445854)

[2.15. Flight 16](#_Toc447445855)

[2.16. Flight Crew 16](#_Toc447445856)

[2.17. Customer 17](#_Toc447445857)

[2.18. Reservation 18](#_Toc447445858)

[2.19. Ticket 19](#_Toc447445859)

[2.20. Promotion 20](#_Toc447445860)

[2.21. Campaign 20](#_Toc447445861)

[2.22. Sale 21](#_Toc447445862)

[3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES 22](#_Toc447445863)

[4. FUNCTIONAL COMPONENTS 23](#_Toc447445864)

[4.1. Use Cases/Scenarios 23](#_Toc447445865)

[4.1.1. Customer Use Cases 24](#_Toc447445866)

[4.1.2. Manager Use Cases 27](#_Toc447445867)

[4.1.3. Salesperson Use Cases 29](#_Toc447445868)

[4.2. Algorithms 31](#_Toc447445869)

[4.2.1. Price Related Algorithms 31](#_Toc447445870)

[4.2.2. Promotion Related Algorithms 31](#_Toc447445871)

[4.2.3. Reservation Related Algorithms 32](#_Toc447445872)

[4.3. Data Structures 32](#_Toc447445873)

[6. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS 33](#_Toc447445874)

[6.1. Customer User Interface Design 33](#_Toc447445875)

[6.1.1. Customer Home Screen 33](#_Toc447445876)

[6.1.2. Customer Search Flight Screen with One Way Ticket 35](#_Toc447445877)

[6.1.3. Result of Flight Search Screen 36](#_Toc447445878)

[6.1.4. Login Screen 37](#_Toc447445879)

[6.1.5. Create Account Screen 38](#_Toc447445880)

[6.1.6. Manage Account Screen 39](#_Toc447445881)

[6.1.7. Result of Flight Search Screen for Logged in Customers 41](#_Toc447445882)

[6.1.8. Customer Reservations & Tickets Screen 43](#_Toc447445883)

[6.2. Manager User Interface Design 48](#_Toc447445884)

[6.2.1. Login Screen 48](#_Toc447445885)

[6.2.2. Manager Manage Account Screen 49](#_Toc447445886)

[6.2.3. Manage Airports Screen 51](#_Toc447445887)

[6.2.4. Manage Airports Screen 53](#_Toc447445888)

[6.2.5. Manage Routes Screen 56](#_Toc447445889)

[6.2.6. Manage Flights Screen 58](#_Toc447445890)

[6.2.7. View Flight Status Screen 65](#_Toc447445891)

[6.2.8. Manage Staff Screens 68](#_Toc447445892)

[6.2.9. View Customers Screen 74](#_Toc447445893)

[5.3. Salesperson User Interface Design 76](#_Toc447445894)

[5.3.1. Salesperson Login Screen 76](#_Toc447445895)

[5.3.2. Salesperson Manage Account Screen 77](#_Toc447445896)

[5.3.3. Salesperson View Flight Status Screen 77](#_Toc447445897)

[5.3.4. Salesperson Customer View 77](#_Toc447445898)

[5.3.5. Salesperson Customer Reservations &Tickets View 79](#_Toc447445899)

[7. ADVANCED DATABASE COMPONENTS 84](#_Toc447445900)

[7.1. Views 84](#_Toc447445901)

[7.1.1. Customer Flight View 84](#_Toc447445902)

[7.1.2. Customer Reservation View 84](#_Toc447445903)

[7.1.3. Customer Flight Details View 84](#_Toc447445904)

[7.1.4. Customer Ticket View 85](#_Toc447445905)

[7.1.5. Manager Customer View 85](#_Toc447445906)

[7.2. Stored Procedures 86](#_Toc447445907)

[7.2.1. Make Reservation Stored Procedure 86](#_Toc447445908)

[7.2.2. Cancel Reservation Stored Procedure 86](#_Toc447445909)

[7.2.3. Purchase Ticket Stored Procedure 86](#_Toc447445910)

[7.2.4. Refund Ticket Stored Procedure 87](#_Toc447445911)

[7.2.5. Delete Airport Procedure 87](#_Toc447445912)

[7.2.6. Delete Route Procedure 87](#_Toc447445913)

[7.2.7. Cancel Flight Procedure 88](#_Toc447445914)

[7.2.8. Send Plane to Repair Stored Procedure 88](#_Toc447445915)

[7.3. Reports 88](#_Toc447445916)

[7.3.1. Total Number of Customers Registered to the System, Total Number of Customers with Reservation, Total Number of Customers with Tickets 88](#_Toc447445917)

[7.3.2. Total Number of Available Flights, Total Number of Current Reservations, Total Number of Purchased Tickets and The Total Amount of Money Spent by the Customers 89](#_Toc447445918)

[7.3.3. Total Number of Employees in Each Role and the Average Salary of Each Role 89](#_Toc447445919)

[7.3.4. Total Number of Tickets and Reservations Associated with Each Flight and the List of All Tickets and Reservations of the Flight 90](#_Toc447445920)

[7.3.5. Total Number of Assigned Routes and Flights to Each Airport 90](#_Toc447445921)

[7.4. Triggers 91](#_Toc447445922)

[7.4.1. Mile Sum Trigger After Purchasing Ticket 91](#_Toc447445923)

[7.4.2. Mile Sum Trigger After Refunding Ticket 91](#_Toc447445924)

[7.4.3. Cancel Reservation Trigger After Refunding Ticket 91](#_Toc447445925)

[7.4.4. Total Money Trigger After Purchasing Ticket 91](#_Toc447445926)

[7.4.5. Total Money Trigger After Refunding Ticket 91](#_Toc447445927)

[7.4.6. Pay Penalty Trigger After Refunding Ticket 92](#_Toc447445928)

[7.4.7. Total Money Trigger After Buying Meal 92](#_Toc447445929)

[7.4.8. Total Money Trigger After Updating Meal 92](#_Toc447445930)

[7.4.9. Total Money Trigger After Registering Extra Luggage 92](#_Toc447445931)

[7.4.10. Total Money Trigger After Cancelling Extra Luggage 92](#_Toc447445932)

[7.5. Constraints 92](#_Toc447445933)

[7.5.1. Minimum Salary Constraint for Staff 93](#_Toc447445934)

[7.5.2. Capacity Constraint for Planes Assigned to Flights 93](#_Toc447445935)

[7.5.3. Simultaneous Flights Constraint for Customer Flights 93](#_Toc447445936)

[7.5.4. Extra Luggage Constraint for Ticket 93](#_Toc447445937)

[7.5.5. Passenger Capacity Constraint for Plane 93](#_Toc447445938)

[7.5.6. Maximum Travel Time Constraint for Plane 93](#_Toc447445939)

[7.5.7. Seat Number Constraint for Flight Class 94](#_Toc447445940)

[7.5.8. Location Constraint for Crew Assignment 94](#_Toc447445941)

[7.5.9. Pilot Number Constraint for Pilot Assignment 94](#_Toc447445942)

[7.5.10. Purchasing Constraint for Customer 94](#_Toc447445943)

[7.5.11. Total Money Constraint for Customer 94](#_Toc447445944)

[7. IMPLEMENTATION PLAN 94](#_Toc447445945)

# REVISED E/R MODEL

## Changes Made in the E/R Diagram

After we received feedback from our assistant, we made the following changes in our E/R model in order to provide a better database structure for our project:

* Instead of making city an attribute, we represented city as an entity. Besides city name, we keep the climate of the city as well.
* We removed the ‘sold’ attribute from the reservation. When a reservation is purchased by the customer a related ticket is created in the system. Instead of a sold attribute we will join the tables when tickets and reservations need to be linked.
* We removed the primary key attribute of user\_name in Reservation authority since we already have a primary key staff\_id in the parent class.
* In order to distinguish subclasses of ReservationAuthority entity, we extended our diagram as follows:
* We added manager\_room info to Manager entity in order to track the room number of the manager.
* We added counter\_id to Salesperson in order to track which counter the salesperson is currently assigned to.
* We added attribute gate\_number to Ticketing/Gate Agent in order to track which gate the employee is currently assigned to.

During the design process we also discovered new attributes and new aspects of the system. We made the following changes to improve the E/R model:

* Most importantly we eliminated Ticketing/Gate Agent from our system since the duties of the agent were already covered by other staff. Hence, we deleted the Ticketing/Gate Agent entity.
* We added price attribute to ticket to track the price that is paid for each ticket.
* We added total\_money attribute to the customer in order to perform purchasing operations.
* We previously had many-to-many relation between ticket/reservation and customer. However, we decided to allow a ticket or reservation to be owned by only one customer. Hence, we made the relations one-to-many instead.
* We decided to identify a flight with date and departure\_time properties along with plane\_name and route\_id attributes.
* We inserted available attribute to plane to indicate whether the plane is currently available for flight or it is unavailable (on repair, etc. ).
* We added delay attribute to flight to track how many minutes of delay the flight has.
* We added penalty\_amount attribute to ticket in order to track the amount of penalty in cancellation.
* We made the relation between promotion and customer one-to-many instead of many-to-many in order to make sure that a promotion belongs to one customer.
* We added business\_price and economy\_price attributes to flight to record prices of flights.
* We added meals and luggage attribute to flight in order to track the meals available for the flight and the total luggage registered.

## Revised E/R Diagram

# RELATION SCHEMAS

## Staff

**Relational Model:**

Staff(staff\_id, name, salary, birthdate, age, gender)

**Functional Dependencies:**

staff\_id ->name, salary, birthdate, age, gender

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

create table staff

(staff\_id int PRIMARY KEY,

name varchar(32) NOT NULL,

salary int NOT NULL,

birthdate date,

age int,

gender char(1) );

## Staff Phones

**Relational Model:**

Staff\_Phones(staff\_id, phone)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(staff\_id, phone)}

**Normal Form:**

BCNF

**Table Definition:**

create table staff\_phones

(staff\_id int PRIMARY KEY,

phone char(15) NOT NULL );

## Reservation Authoritative

**Relational Model:**

Reservation\_Authoritative (staff\_id, user\_name, password)

**Functional Dependencies:**

staff\_id -> user\_name, password

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

create table reservation\_authoritative

(staff\_id int PRIMARY KEY,

user\_name varchar(32) NOT NULL,

password varchar(32) NOT NULL,

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## Permissions

**Relational Model:**

Permissions (staff\_id, permission\_name)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(staff\_id, permission\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE permissions

(staff\_id int PRIMARY KEY,

permission\_name varchar(32),

FOREIGN KEY (staff\_id) REFERENCES reservation\_authoritative ) ENGINE = InnoDB;

## Manager

**Relational Model:**

Manager(staff\_id, manager\_room)

**Functional Dependencies:**

staff\_id -> manager\_room

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE manager

(staff\_id int PRIMARY KEY,

manager\_room varchar(4)

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## Salesperson

**Relational Model:**

Salesperson(staff\_id, counter\_id)

**Functional Dependencies:**

staff\_id -> counter\_id

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE salesperson

(staff\_id int PRIMARY KEY,

counter\_id int

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## Crew

**Relational Model:**

Crew(staff\_id, license\_no, rank, current\_location)

**Functional Dependencies:**

staff\_id -> license\_no, rank, current\_location

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE crew

(staff\_id int PRIMARY KEY,

license\_no char(10) NOT NULL,

rank int NOL NULL,

current\_location varchar(16) NOT NULL,

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## Pilot

**Relational Model:**

Pilot(staff\_id, max\_flight\_distance)

**Functional Dependencies:**

staff\_id -> max\_flight\_distance

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE pilot

(staff\_id int PRIMARY KEY,

max\_flight\_distance int NOT NULL

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## Flight Attendance

**Relational Model:**

Flight\_Attendance(staff\_id, flight\_class\_served)

**Functional Dependencies:**

staff\_id -> flight\_class\_served

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE flight\_attendance

(staff\_id int PRIMARY KEY,

flight\_class\_served char(1) NOT NULL,

FOREIGN KEY (staff\_id) REFERENCES staff ) ENGINE = InnoDB;

## City

**Relational Model:**

City(city\_name, climate)

**Functional Dependencies:**

city\_name -> climate

**Candidate Keys:**

{(city\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE city

(city\_name varchar(20) PRIMARY KEY,

climate char(1));

## Airport

**Relational Model:**

Airport(airport\_id, airport\_capacity, address, city\_name)

**Functional Dependencies:**

airport\_id -> airport\_capacity, address, city\_name

**Candidate Keys:**

{(airport\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE airport

(airport\_id int PRIMARY KEY,

airport\_capacity int NOT NULL,

address varchar(100),

city\_name varchar(20),

foreign key (city\_name) REFERENCES city) ENGINE InnoDB;

## Route

**Relational Model:**

Route(route\_id, flight\_duration, departs, arrives)

**Functional Dependencies:**

route\_id -> flight\_duration, departs, arrives

**Candidate Keys:**

{(route\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE route

(route\_id int PRIMARY KEY,

flight\_duration time NOT NULL,

departs int NOT NULL,

arrives int NOT NULL,

FOREIGN KEY (departs) REFERENCES airport,

FOREIGN KEY (arrives) REFERENCES airport) ENGINE InnoDB;

## Plane Type

**Relational Model:**

Plane\_Type(plane\_type\_id, seat\_map, capacity, max\_flight\_time, plane\_model, brand, storage\_capacity, production\_year, width, height, length)

**Functional Dependencies:**

plane\_type\_id -> seat\_map, capacity, max\_flight\_time, plane\_model, brand, storage\_capacity, production\_year, width, height, length

**Candidate Keys:**

{(plane\_type\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE plane\_type

(plane\_type\_id varchar(4) PRIMARY KEY,

seatmap varchar(200) NOT NULL,

capacity int NOT NULL,

max\_flight\_time time NOT NULL,

plane\_model varchar(4),

brand varchar(10),

storage\_capacity int NOT NULL,

production\_year int,

width int,

height int,

length int );

## Plane

**Relational Model:**

Plane(plane\_name, available, plane\_type\_id)

**Functional Dependencies:**

plane\_name -> available, plane\_type\_id

**Candidate Keys:**

{(plane\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE plane

(plane\_name varchar(20) PRIMARY KEY,

available char(1) NOT NULL,

FOREIGN KEY (plane\_type\_id) REFERENCES plane\_type) ENGINE InnoDB;

## Flight

**Relational Model:**

Flight(plane\_name, route\_id, date, departure\_time, flight\_id, delay, business\_price, economy\_price, meals, luggage)

**Functional Dependencies:**

plane\_name, route\_id , date, departure\_time -> flight\_id, delay, business\_price, economy\_price , meals, luggage

**Candidate Keys:**

{(plane\_name, route\_id, date, departure\_time)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE flight

(plane\_name varchar(20) PRIMARY KEY,

route\_id int PRIMARY KEY,

date date NOT NULL,

departure\_time time NOT NULL,

flight\_id int NOT NULL,

delay int,

business\_price int NOT NULL,

economy\_price int NOT NULL,

meals varchar(50),

luggage int,

FOREIGN KEY (route\_id) REFERENCES route,

FOREIGN KEY (plane\_name) REFERENCES plane) ENGINE = InnoDB;

## Flight Crew

**Relational Model:**

Flight\_Crew(staff\_id, plane\_name, route\_id, date, departure\_time)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(plane\_name, route\_id, date, departure\_time)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE flight\_crew

(staff\_id int PRIMARY KEY,

plane\_name varchar(20) PRIMARY KEY,

route\_id int PRIMARY KEY,

date date NOT NULL,

departure\_time time NOT NULL,

FOREIGN KEY (staff\_id) REFERENCES staff,

FOREIGN KEY (plane\_name, route\_id, date, departure\_time) REFERENCES flight) ENGINE = InnoDB;

## Customer

**Relational Model:**

Customer(user\_name, password, name, birthdate, age, passport\_no, mile\_sum, total\_money)

**Functional Dependencies:**

user\_name -> password, name, birthdate, age, passport\_no, mile\_sum, total\_money

**Candidate Keys:**

{(user\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLEcustomer

(user\_name varchar(32) PRIMARY KEY,

password varchar(32) NOT NULL,

name varchar(32) NOT NULL,

birthdate date,

age int,

passport\_no char(9),

mile\_sum int

totalMoney int NOT NULL);

## Reservation

**Relational Model:**

Reservation(user\_name, plane\_name, route\_id, date, departure\_time, reservation\_no, class, cancelled)

**Functional Dependencies:**

user\_name, plane\_name, route\_id, date, departure\_time, reservation\_no -> class, cancelled

**Candidate Keys:**

{(user\_name, plane\_name, route\_id, date, departure\_time, reservation\_no)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE reservation

(user\_name varchar(32) PRIMARY KEY,

plane\_name varchar(20) PRIMARY KEY,

route\_id int PRIMARY KEY,

reservation\_no int PRIMARY KEY,

date date NOT NULL,

departure\_time time NOT NULL,

class char(1) NOT NULL,

cancelled char(1) NOT NULL,

FOREIGN KEY (user\_name) REFERENCES customer,

FOREIGN KEY (plane\_name, route\_id, date, departure\_time) REFERENCES flight) ENGINE = InnoDB;

## Ticket

**Relational Model:**

Ticket( user\_name, plane\_name, date, departure\_time, route\_id, ticket\_no, extra\_luggage, meal, class, penalty\_amount, seat\_no, price )

**Functional Dependencies:**

user\_name, plane\_name, route\_id, date, departure\_time, ticket\_no -> extra\_luggage, meal, class, penalty\_amount, seat\_no, price

**Candidate Keys:**

{(user\_name, plane\_name, route\_id, date, departure\_time, ticket\_no)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE ticket

(user\_name varchar(32) primary key,

plane\_name varchar(20) PRIMARY KEY,

route\_id int PRIMARY KEY,

date date NOT NULL,

departure\_time time NOT NULL,

ticket\_no int PRIMARY KEY,

price int NOT NULL,

class char(1) NOT NULL,

meal varchar(20),

extra\_luggage int,

seat\_no varchar(4),

FOREIGN KEY (user\_name) REFERENCES customer,

FOREIGN KEY (plane\_name, route\_id, date, departure\_time) REFERENCES flight) ENGINE = InnoDB;

## Promotion

**Relational Model:**

Given\_Promotion(user\_name, promotion\_id)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**CREATE TABLE** given**\_**promotion

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

FOREIGN KEY user\_name REFERENCES customer);

## Campaign

**Relational Model:**

Campaign(user\_name, promotion\_id, campaign\_type)

**Functional Dependencies:**

user\_name, promotion\_id -> campaign\_type

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**create table** campaign

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

campaign\_type varchar(10) NOT NULL,

FOREIGN KEY(user\_name, promotion\_id) references given\_promotion);

## Sale

**Relational Model:**

Sale(user\_name, promotion\_id, sale\_amount, sale\_period)

**Functional Dependencies:**

user\_name, promotion\_id -> sale\_amount, sale\_period

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**CREATE TABLE** sale

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

sale\_amount int,

sale\_period interval,

FOREIGN KEY (user\_name, promotion\_id) REFERENCES given\_promotion);

# FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

In Relation Schemas part of the report, the normal form of all tables are indicated. Since all the relations are either in BCNF or 3NF form no decomposition or further normalization was needed.

# FUNCTIONAL COMPONENTS

## Use Cases/Scenarios

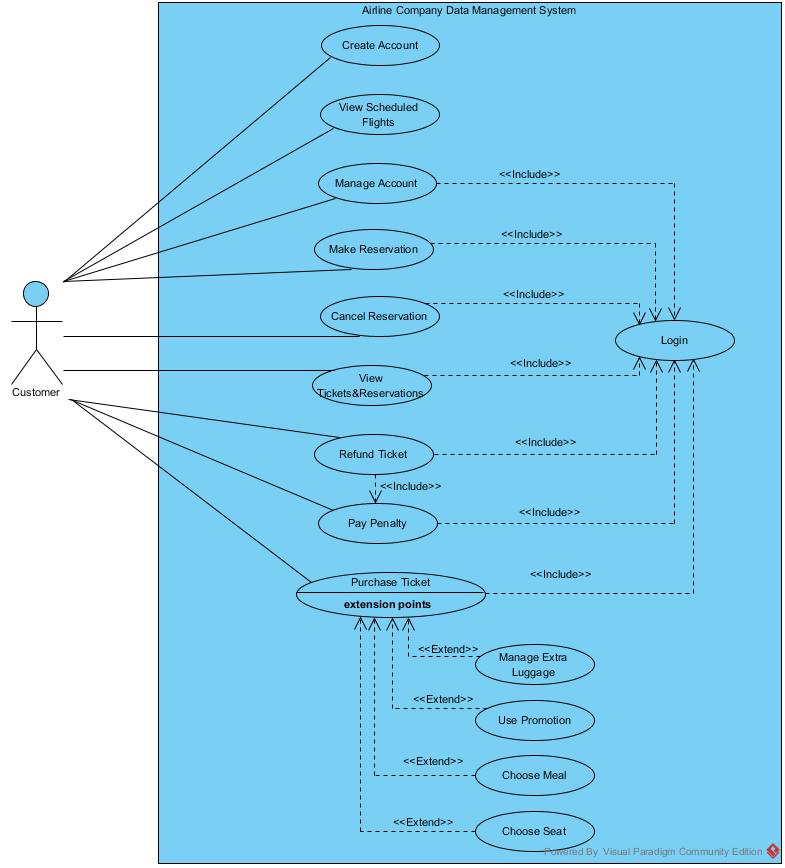
Airline Company Data Management System is responsible from providing reservation services to users along with allowing related managers to control flight, employee, airport, and flight details. The service details of the system vary according to the user. Airline Company Data Management System has 3 users: Customer, Manager, and Salesperson. Even though some services are common to all users, each user is allowed to access different functionalities of the system. Ticketing/Gate Agent user was removed from the system since it was seen unnecessary.

Customer is able to perform actions related to reservations such as viewing flight details and making payment.

Manager is the admin of the system and can manage planes, flights, airports, reservations, crew, and staff.

Salesperson is responsible from helping the customer to complete the reservation and purchasing services.

### Customer Use Cases

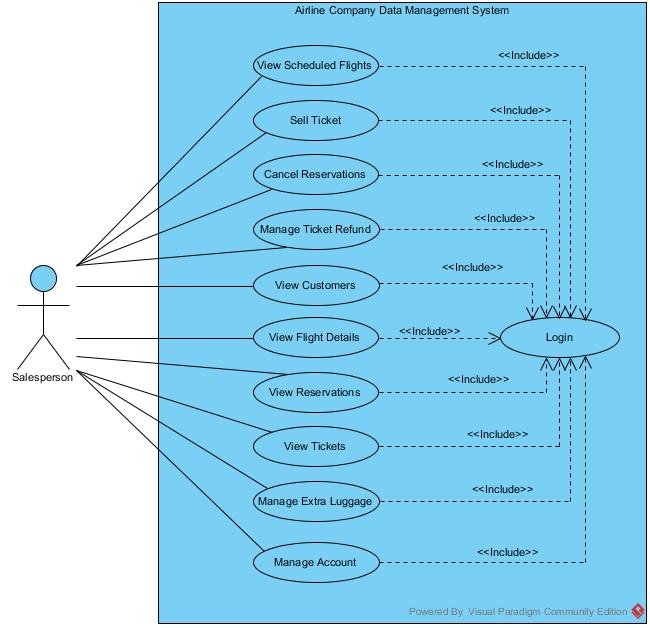


* **Login:** The customer can login to the system using his/her username and password in order to be able make reservation/purchasing operations. Except create account and view scheduled flight use cases, login is required before all operations.
* **Create Account:** The customer can create an account in the system by specifying his/her unique username, password, name, birthdate, and passport number. The user can use this account information to login to the system for reservations. The total miles travelled and the account balance is set to 0 automatically.
* **Manage Account:** The customer can update his/her account information. The username, name, password, birthday, passport details can be updated by the customer when it is necessary.
* **View Scheduled Flights:** The customer can view all flights that are registered to system. He/she can specify date, arrival city and airport, and departure city and airport in order to view the flights he/she is interested in. The customer can also search for one-way or return flights.
* **Make Reservation:** The customer can make reservations to the flights they select from the list of available flights.
* **Cancel Reservation:** The customer can cancel their reservations if the system allows them to. He/she can select the reservation to cancel from the list of his/her reservations.
* **View Tickets & Reservations:** The customer can view all his/her reserved flights and purchased flights. Past and cancelled flights can also be viewed. The reserved flight information includes departure/arrival airports, date, departure time, price, and class details. The departure/arrival airports, date, departure time, meals, class, seat number, extra luggage, price details are visible for the purchased flights. The flight selections such as meal and luggage can be updated by the customer.
* **Refund Ticket:** The customer can refund his/her tickets if the system allows by selecting the flight from the list of his/her flights. He/she can cancel his/her ticket and the money will be refunded. The Refund Ticket use case includes paying the penalty amount for cancelling the ticket.
* **Pay Penalty:** The customer can pay the penalty amount from his/her account when a ticket is refunded. The amount of the penalty will be deducted from the customer account.
* **Purchase Ticket:** The customer can purchase flights that exist in the list of available flights or the flights that he/she reserved. Purchase ticket use case can also cover adding/cancelling extra luggage, selecting promotion to use, choosing meal and seat number.
* **Manage Extra Luggage:** The customer can buy extra luggage for the flights they bought a ticket from. They can indicate the extra luggage amount and make the necessary payment. The customer can also cancel the bought extra luggage. In this case their money will be refunded. Managing extra luggage can be done during purchasing the ticket or after the purchase.
* **Use Promotion:** The customer can benefit from the given promotions while purchasing a flight. The list of all campaigns and sales available are shown to user during purchase operation. The customer can select a sale or campaign to use, and the price of the ticket is updated accordingly.
* **Choose Meal:** The customer can choose meal for the purchased flight and make the necessary payment. The meal selection can be updated after the purchase if the system allows the operation.
* **Choose Seat:** The customer can choose his/her seat for the flights they bought a ticket from if the seat map information is available. The seat options are determined according to the flight class. The seat number can be updated after the purchase if the system allows the operation.

### Manager Use Cases

* **Login:** Manager can login to the system with his/her username and password. The manager will be given access to manager functions when the account details are approved. Manager needs to login to the system to be able to perform other operations.
* **Manage Airports:** The manager can view all airports registered to the system and their details: capacity, address, city, id, assigned routes and flights. He/she can register new airports to the system and delete existing airports. The manager can also change details of the airports such as airport capacity.
* **Manage Planes:** The manager can view all the planes registered to the system. He/ she can register new planes to the system by specifying the id and the plane type and delete the existing ones. Furthermore, the manager can send the planes to repair and update the status of the planes as unavailable. He/she can also mark the planes returned from repair as available.
* **Manage Routes:** The manager can view all routes registered to the system. He/she can add new routes by specifying the route id, source and destination airports, and the flight time. The manager is also able to delete the existing routes. The details of the routes can also be changed, the flight time and associated airports can be modified.
* **Manage Flights:** The manager can view all current flights in the system with airport, time, status, crew details. He/she can add new flights by specifying date, time, flight, route details. The manager can also delete or cancel the existing flights. Furthermore, he/she can change the details of the flights, changed assigned planes, modify date, time or route details.
* **Manage Crew Assignment:** The manager can assign crew to the flights according to the date and flight time of the flight and location and rank of the crew. The manager is able to assign both pilots and flight attendance. He/she can also take back the assigned crew from flights.
* **Manage Staff:** The manager is able to see all details of the information of staff: pilots, flight attendance, salespersons, and ticketing/gate agents. The manager is able to hire new staff by entering id, name, salary, birthday, gender, phone, license, duty details. He/she can also delete the existing staff, fire them. The manager is also able to change the information of the staff, raise or lower their salaries, update their licenses, change their flight distance and flight class.
* **View Reservations:** The manager is able to view all reservations and view the associated customer, date, route, plane, and, class details.
* **View Tickets:** The manager is able to view all tickets and view the associated customer, date, route, plane, crew, meal, seat, luggage and class details.
* **Manage Account:** The manager is able to view the details of his account, see username, password details. He/she can also update his account, change username/password, phone details.

### Salesperson Use Cases



* **Login:** Salesperson can login to the system with his/her username and password. The salesperson will be given access to salesperson functions when the account details are approved. The salesperson is required to login to the system in order to be able to perform all operations.
* **View Scheduled Flights:** The salesperson can view all flights that are registered to system. He/she can specify date, arrival city and airport, and departure city and airport in order to help the customers to make reservations.
* **Sell Ticket:** The salesperson can complete the ticket purchasing operations for the customers. From the customer reservations, the salesperson can complete the purchasing operations. The salesperson can also select the ticket details such as meal, class, seat number for the customer.
* **Cancel Reservation:** The salesperson can cancel customer reservations when necessary by deleting the reservation from the system.
* **Manage Ticket Refund:** The salesperson can refund the tickets of the customers when necessary. He/she can cancel the ticket –delete it from the system, refund the money to the customer account and also charge the penalty.
* **View Customers:** The salesperson can view the list of customers and choose the customer to handle reservation and purchase operations.
* **Manage Account:** The salesperson is able to view the details of his account, see username, password details. He/she can also update his account, change username/password, phone details.
* **View Flight Details:** The salesperson can view the status of all existing flights including their delay amount, class options and available meals.
* **View Reservations:** The salesperson is able to view all reservations of a customer and view date, time, route, and, class details.
* **View Tickets:** The salesperson is able to view all tickets of a customer and view the associated customer, date, route, plane, crew, meal, seat, luggage and class details.
* **Manage Extra Luggage:** The salesperson can register extra luggage for the customers. He/she can also make payment operations for the customer in order to buy extra luggage. The manager is also able to cancel the bought extra luggage and refund the money back to customers.

## Algorithms

### Price Related Algorithms

Every customer interacting with the application has to buy a ticket in order to have a trip. In our system, the ticket information is kept under Ticket table and this entity has an important role while maintaining the system. Since buying a ticket will affect Customer’s attributes directly, mile\_sum and total\_amount, it is very crucial to hold Ticket’s price.

Prices of the tickets will be set according to their flight locations by default. However, as the time passes, their prices will change regarding the days left until the flight.

Flights’ prices, which was set by default in the beginning, will stay in their first amounts until there are two weeks until the flight. At this time, the price will be increased by 25%. After a week, the price will be once again increased by 25%. For instance, if a flight between Ankara and Istanbul is set to be 100TL at the beginning, its price will be 125TL when there are two weeks until the flight time. A week later, when there is a week left until the flight time, its price will be 156.25TL and the price will change no more. The prices are different for the classes. All initial prices are set by the manager.

Meal and extra\_luggage prices are the same for each flight and for all times. Whenever the user selects meal and luggage the price of the ticket is recalculated.

The penalty\_amount associated with each flight follows the same algorithm with the ticket price. The penalty\_amount will be increased 25% in the last 2 weeks and will be increased 25% again in the last week.

### Promotion Related Algorithms

The system will have promotions to encourage customers to tickets more frequent or to offer them ticket prices in a reasonable amount. Since the promotions are distinguished with a primary key of promotion\_id, no other customer than the owner of the promotion will be able to use it.

Promotions are divided into two other entities which are defined by disjoint specialization. First one, Campaign, rewards the customer with a free ticket depending on the mile\_sum the customer has. For customers who have a total of 10,000 miles, a free one-way ticket will be given to be used only inside Europe. For a total of 25,000 miles, the customer will be given a one-way overseas flight.

Other type of Promotion, Sale, provides the customer with a percentage of discount under a specific time period. The discount of 15% will be given on customer’s birthday within a time period of one week. Moreover, for each customer, for every 5,000 miles they gain, they will be given a discount of 20% in a time period of two weeks.

The flight history of the customer also affects the promotions. For every 3 cancellations one future or available sale or promotion is cancelled.

### Reservation Related Algorithms

Customers can reserve their tickets with a unique reservation\_no. They are able to choose the class of their ticket as well. However, if they cancel their flights, then there is no returning back. They will no longer be able to have their reservation turned into a ticket.

In order to prevent customers making more reservations then they need, they will be given 2 days to buy their tickets or else, the price of their tickets will be increased by 10%. In this way, they will be deterred from occupying tickets. Moreover, if the reservation is not turned into ticket in 24 hours before the flight, the reservation is cancelled directly.

## Data Structures

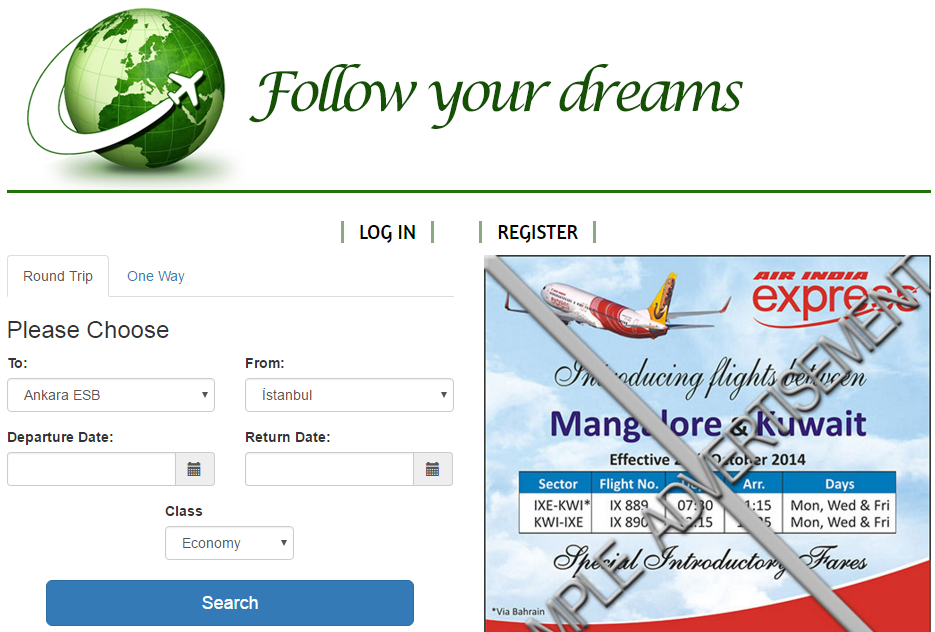
In our relation schemas we use Numeric type, String type, Time type, Date type, and Interval type. String type is required to store any character-composed attributes such as names, ids, addresses, phones. Attributes with Numeric domain is used in order to store numeric data such as age, capacity, mile sum. Time type is used to keep time interval values such as flight time. Date type is used to specify birthdates, flight dates and any other day specifications. The Interval domain was necessary to keep track of the sale period.

# USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

## Customer User Interface Design

The screens customer can access are described in detail.

### Customer Home Screen

****

**Inputs:** @to, @from, @departuredate, @arrivaldate, @class

**Process:** The homepage of the Airline Company Data Management system is in the above figure. When the login button is clicked the user is directed to the login screen. Register button allows the customer to create an account in the system. Login is not necessary in order to search for flights. In order to search for flights for a round trip, the user needs to select airport-city pair for the to and from fields from the list of available airport-city pairs. The user also selects the departure date and return date. The class is also selected by the user. When the select button is clicked all matching flights are displayed in the available flights page.

**SQL Statements:**

**Listing the Airports**

SELECT airport\_name, city\_name

FROM airport

**Searching Matching Flights:**

(Lists all matching flights that are within 2 days after and before user selected dates)

( SELECT flight\_id, date, departure\_time,

CASE WHEN @class = ‘Business’ business\_price

ELSE economy\_price

END

FROM CustomerFlightView C

WHERE departs = @to AND

arrives = @from AND

depatureDate = between @departuredate - 2 and @departuredate + 2) UNION

( SELECT flight\_id, date, departure\_time, price

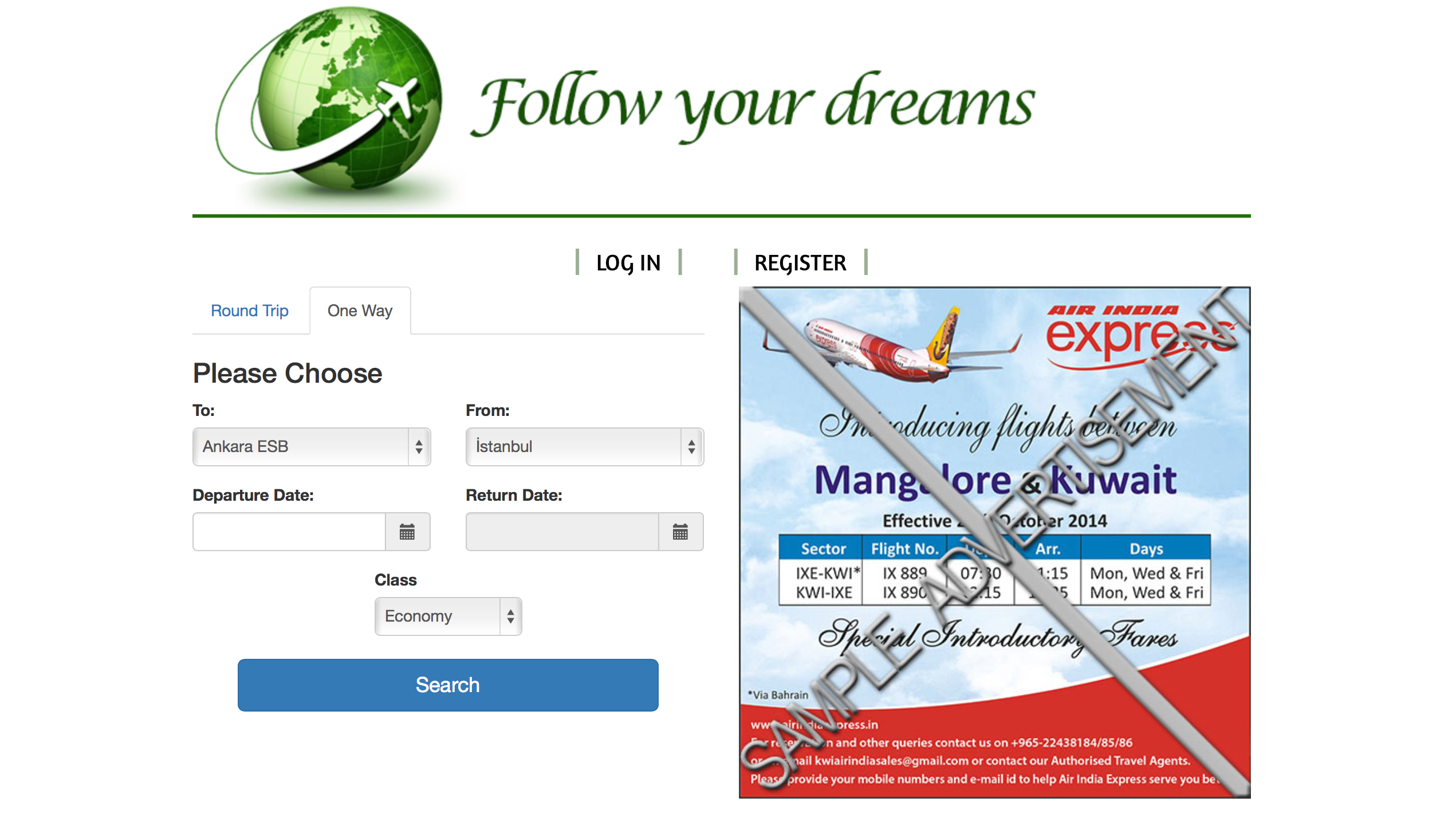
FROM CustomerFlightView C

WHERE departs = @from AND

arrives = @to AND

depatureDate = between @arrivaldate - 2 and @arrivaldate + 2 )

### Customer Search Flight Screen with One Way Ticket



**Inputs:** @to, @from, @departuredate, @class

**Process:** In order to search for flights for one way flights, the user first selects the One Way tab. Then, he/she selects airport-city pair for the to and from fields from the list of available airport-city pairs. The user also selects the departure date and the class. When the select button is clicked all matching flights are displayed in the available flights page.

**SQL Statements:**

**Listing the Airports**

SELECT airport\_name, city\_name

FROM airport

**Searching Matching Flights:**

SELECT flight\_id, date, departure\_time,

CASE WHEN @class = ‘Business’ business\_price

ELSE economy\_price

END

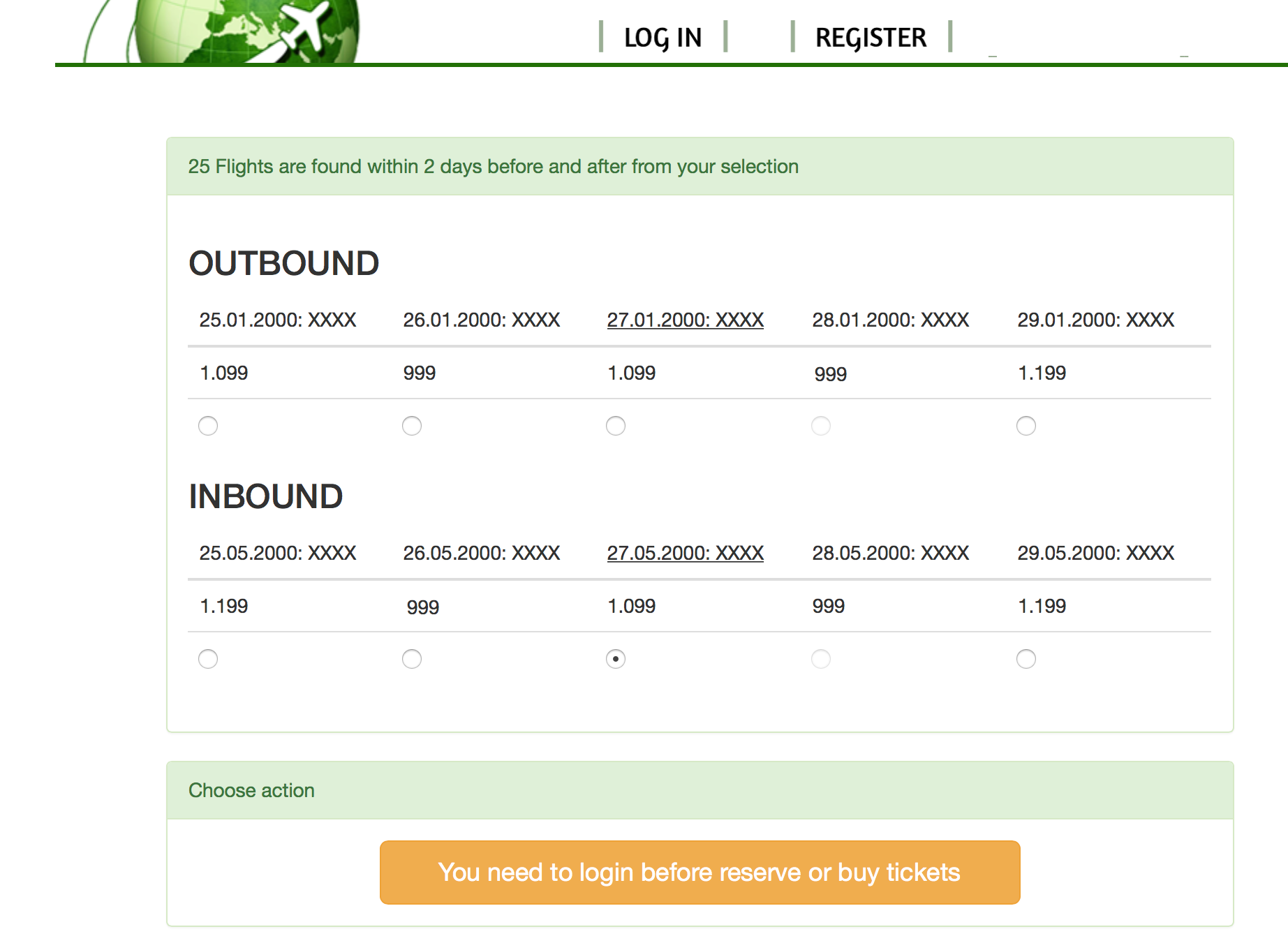
FROM CustomerFlightView C

WHERE departs = @to AND

arrives = @from AND

depatureDate = between @departuredate - 2 and @departuredate + 2

### Result of Flight Search Screen



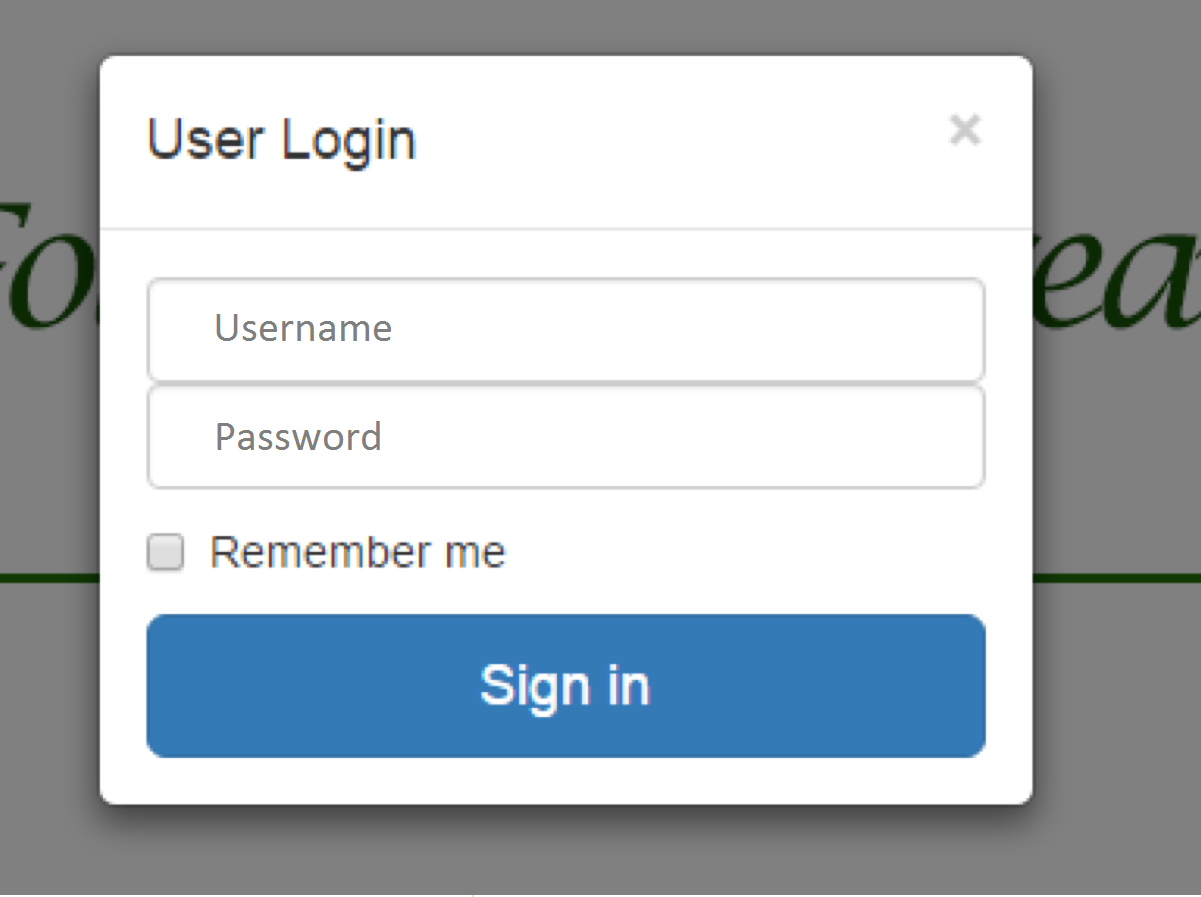
**Inputs:** No input since without login no further operation can be done.

**Process:** When the user enters flight details in pages 5.1.1. and 5.1.2. and clicks ‘Search’ button the list of available flights are displayed to him/her. As shown in the above page, the departure tickets are listed under ‘Outbound’ and the arrival tickets are listed under ‘Inbound’. The number of available flights are also shown to user. The date, time and price of each flight is shown in detail where price is shown with respect to the previously selected class in search. The customers are able to search for flights; however, they are required to log in to the system to be able make reservations or buy tickets.

**SQL Statements:**

The result of SQL statements in 5.1.1. and 5.1.2. are shown in this page. Flights page do not have additional queries.

### Login Screen



**Inputs:** @username, @password

In the homepage, a pop-up window box appears when “Login” is clicked. The box will be asking for access details in order to proceed. Users will be distinguished according to the details they have entered.

**SQL Statements:**

**Login Operation**

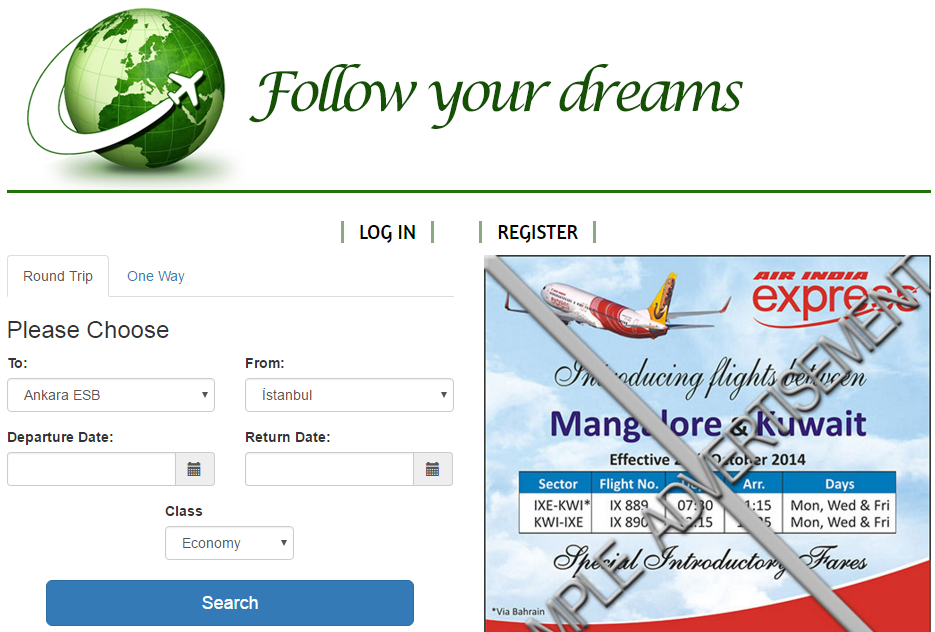
SELECT \*

FROM customer

WHERE user\_name = @username AND

password = @password

### Create Account Screen



**Inputs:** @username, @password, @passwordagain, @name, @birthdate, @passportno

**Process:** The customer can view create account page after clicking on “Register” button from the main menu. He/she enters username, password, name, birthday and passport no information. When “Save Changes” is clicked the account would be created.

**SQL Statements:**

**Create Account**

INSERT INTO customer

VALUES(@username, @password, @name, @birthdate, GETDATE() - @birthdate, @passportno, 0, 0)

WHERE @password = @passwordagain

### Manage Account Screen

****

**Inputs:** @username, @newusername, @password, @passwordagain, @name, @birthdate, @passportno

**Process:** In this page that is provided for customers, customers are able to change their passwords, change the possible mistakes in their names, or in their birthdates and in their passport numbers. Lastly, they are shown their current mile sum so that they are informed about their situation and they can use them accordingly to get a cheaper/free ticket depending on their promotions.

**SQL Statements:**

**View Customer Details**

SELECT \*

FROM customer

WHERE user\_name = @username

**Change Username**

UPDATE customer

SET user\_name = @newusername

WHERE user\_name = @username

**Change Password**

UPDATE customer

SET password = @password

WHERE user\_name = @username AND  
 @password = @passwordAgain

**Change Name**

UPDATE customer

SET name = @name

WHERE user\_name = @username

**Change Birthdate**

UPDATE customer

SET birthdate = @birthdate

WHERE user\_name = @username

**Change Passport Number**

UPDATE customer

SET passport\_no = @passportno

WHERE user\_name = @username

### Result of Flight Search Screen for Logged in Customers



**Inputs:** Inputs: @to, @from, @departuredate, @departuretime, @class, @username

**Process:** When the user enters flight details in pages 5.1.1. and 5.1.2. and clicks ‘Search’ button the list of available flights are displayed to him/her. As shown in the above page, the departure tickets are listed under ‘Outbound’ and the arrival tickets are listed under ‘Inbound’. The number of available flights are also shown to user. The date, time and price of each flight is shown in detail where price is shown with respect to the previously selected class in search. The customers are able to search for flights and the flight they want from the list. ‘Reserve Selection’ button is clicked to reserve the selected flight. ‘Purchase Selection’ is used to directly buy the flight without prior reservation. If the customer searches for a one-way flight in the search flight screen only outbound flights are shown to the customer.

**SQL Statements:**

The result of SQL statements in 5.1.1. and 5.1.2. are shown in this page.

**Make Reservation**

INSERT INTO reservation

FROM flight NATURAL JOIN route

SELECT (user\_name, plane\_name, route\_id, date, departure\_time, INDEX(reservation\_no), @class, 0 )

WHERE user\_name = @username AND

route\_id in (SELECT route\_id FROM route where departs = @from and arrives = @to) AND

date = @departuredate AND

time = @departuretime

**Purchase Ticket**

INSERT INTO ticket

FROM flight NATURAL JOIN route

SELECT (user\_name, plane\_name, route\_id, date, departure\_time, INDEX(ticket\_no), 0, NULL, @class, 0, NULL, CASE WHEN @class = ‘Business’ business\_price ELSE economy\_price END)

WHERE user\_name = @username AND

route\_id in (SELECT route\_id FROM route where departs = @from and arrives = @to) AND

date = @departuredate AND

time = @departuretime

Executes 6.4.1. Mile Sum Trigger After Purchasing Ticket

Executes 6.4.4. Total Money Trigger After Purchasing Ticket

### Customer Reservations & Tickets Screen



**Inputs:** @username (passed from login session), @reservationno, @ticketno, @selectedPromotion, @extraluggage, @meal, @class, @seat\_no

After clicking the Reservations tab after logging in, the page above is shown. The customers can see their existing reservations in detail. When they click “Details” button, they are able to see their ticket’s every single available information. Beside these, they can also change their preferences on their tickets. On the upper right corner of every single reservation, they have the options of changing their meal choice, buying/cancelling extra luggage and changing seats within the same class, if there are seats available. Lastly, they can return their tickets with the red button provided. The page lists both reserved and purchased flights. Also past flights are listed at the bottom of the page. For reservations ‘Cancel’ and ‘Buy’ buttons are available. The customer clicks ‘Cancel’ button to cancel the reservation and clicks ‘Buy’ button to purchase the ticket. The operations available for already purchased tickets are offered during purchase as well.

For tickets, ‘Apply Promotion’, ‘Change Meal’, ‘Buy Extra Luggage’, ‘Cancel Extra Luggage’, ‘Choose Seat’, ‘Change Seat’, ‘Buy Meal’, ‘Return’ buttons are available. The existence of these buttons depend on the status of the ticket. For example, if the meal is not bought for flight, ‘Buy Meal’ button appears but if the meal is already selected ‘Change Meal’ button is shown instead.

Customer clicks ‘Apply Promotion’ button to select a campaign or sale to apply to the ticket from the list of available promotions. Customer clicks ‘Buy Meal’ to select a meal and ‘Change Meal’ to update the meal choice. He/she clicks ‘Buy Extra Luggage’ to specify the amount of extra luggage and clicks ‘Cancel Extra Luggage’ to delete the existing luggage. The customer clicks ‘Choose Seat’ to select a seat from the list of seats and clicks ‘Change Seat’ to update selection. When customer clicks ‘Return’ button the ticket is refunded.

**SQL Statements:**

**View all Reservations**

SELECT date, time, departs, arrives, flight\_duration, class

FROM CustomerReservationView

**View all Tickets**

SELECT date, time, departs, arrives, flight\_duration, extra\_luggage, class

FROM CustomerTicketView

**View Reservation Details**

SELECT \*

FROM CustomerReservationView

**View all Tickets**

SELECT \*

FROM CustomerTicketView

**Cancel Reservation**

UPDATE reservation

SET cancelled = 1

WHERE reservation\_no = @reservationno AND  
 user\_name = @username

**Return and Refund Ticket**

DELETE FROM ticket

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

Executes 6.4.2. Mile Sum Trigger After Refunding Ticket

Executes 6.4.3. Cancel Reservation Trigger After Refunding Ticket

Executes 6.4.5. Total Money Trigger After Refunding Ticket

Executes 6.4.6. Pay Penalty Trigger After Refunding Ticket

**List Promotions**

SELECT \*

FROM promotion

WHERE user\_name = @username

**Apply Promotion**

UPDATE ticket

SET price = CASE

WHEN @selectedPromotion in (SELECT promotion\_id FROM sale WHERE user\_name = @username) THEN price \* sale\_amount

WHEN @selectedPromotion in (SELECT promotion\_id FROM campaign WHERE user\_name = @username) THEN 0

ELSE price

END

**Buy/Change Meal**

UPDATE ticket

SET meal = @meal

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

Executes 6.4.7. Total Money Trigger After Buying Meal

**Choose/Change Seat**

UPDATE ticket

SET seat\_no = @seatno

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

**Register Extra Luggage**

UPDATE ticket

SET extra\_luggage = @extraluggage

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

**Cancel Extra Luggage**

UPDATE ticket

SET extra\_luggage = 0

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

**Purchase Ticket**

INSERT INTO ticket

FROM flight NATURAL JOIN route

SELECT (user\_name, plane\_name, route\_id, date, departure\_time, INDEX(ticket\_no), 0, NULL, @class, 0, NULL, CASE WHEN @class = ‘Business’ business\_price ELSE economy\_price END)

WHERE user\_name = @username AND

route\_id in (SELECT route\_id FROM route where departs = @from and arrives = @to) AND

date = @departuredate AND

time = @departuretime

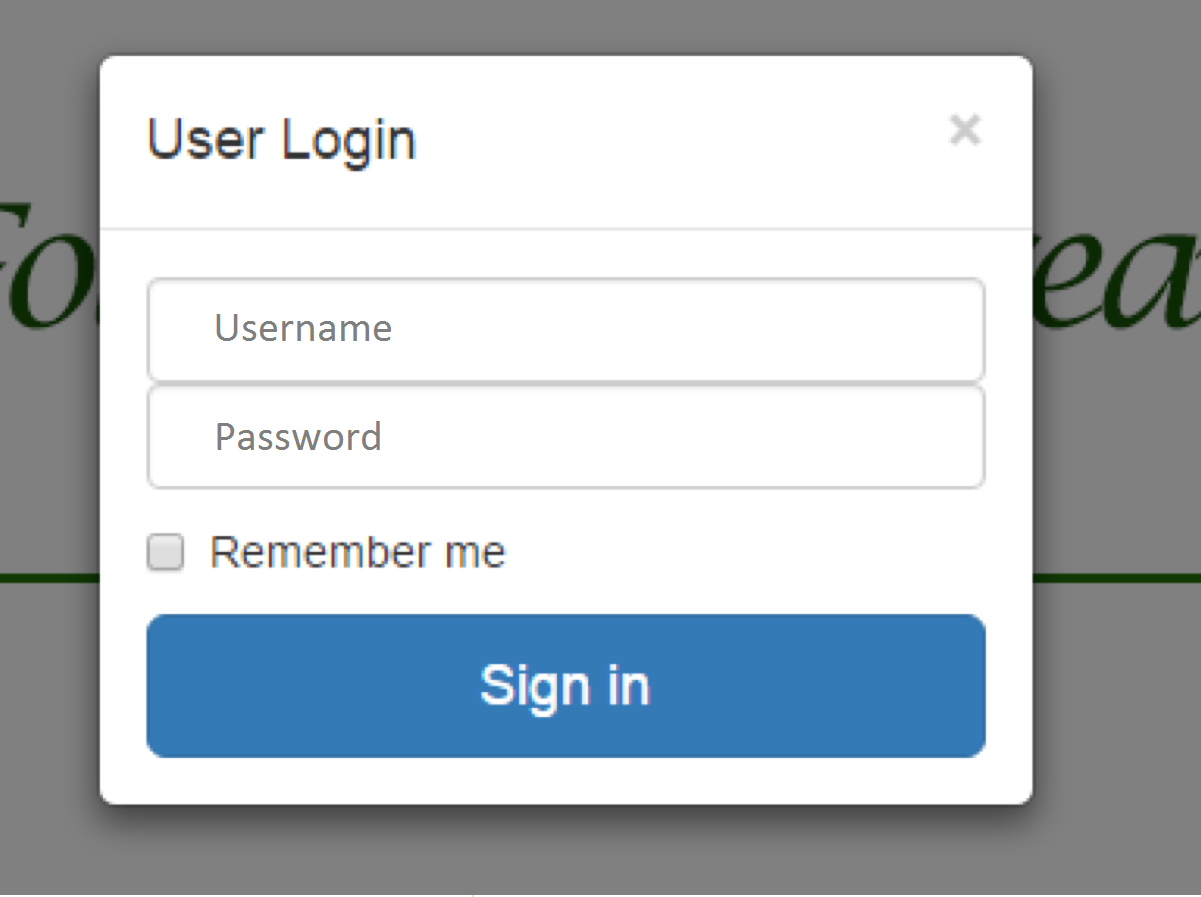
Executes 6.4.1. Mile Sum Trigger After Purchasing Ticket

Executes 6.4.4. Total Money Trigger After Purchasing Ticket

## Manager User Interface Design

The screens manager can access are described in detail.

### Login Screen



**Inputs:** @username, @password

**Process:** The manager clicks ‘Login’ button. He/she enters his/her username and password in order to access to manager operations.

**SQL Statements:**

**Login to System**

SELECT \*

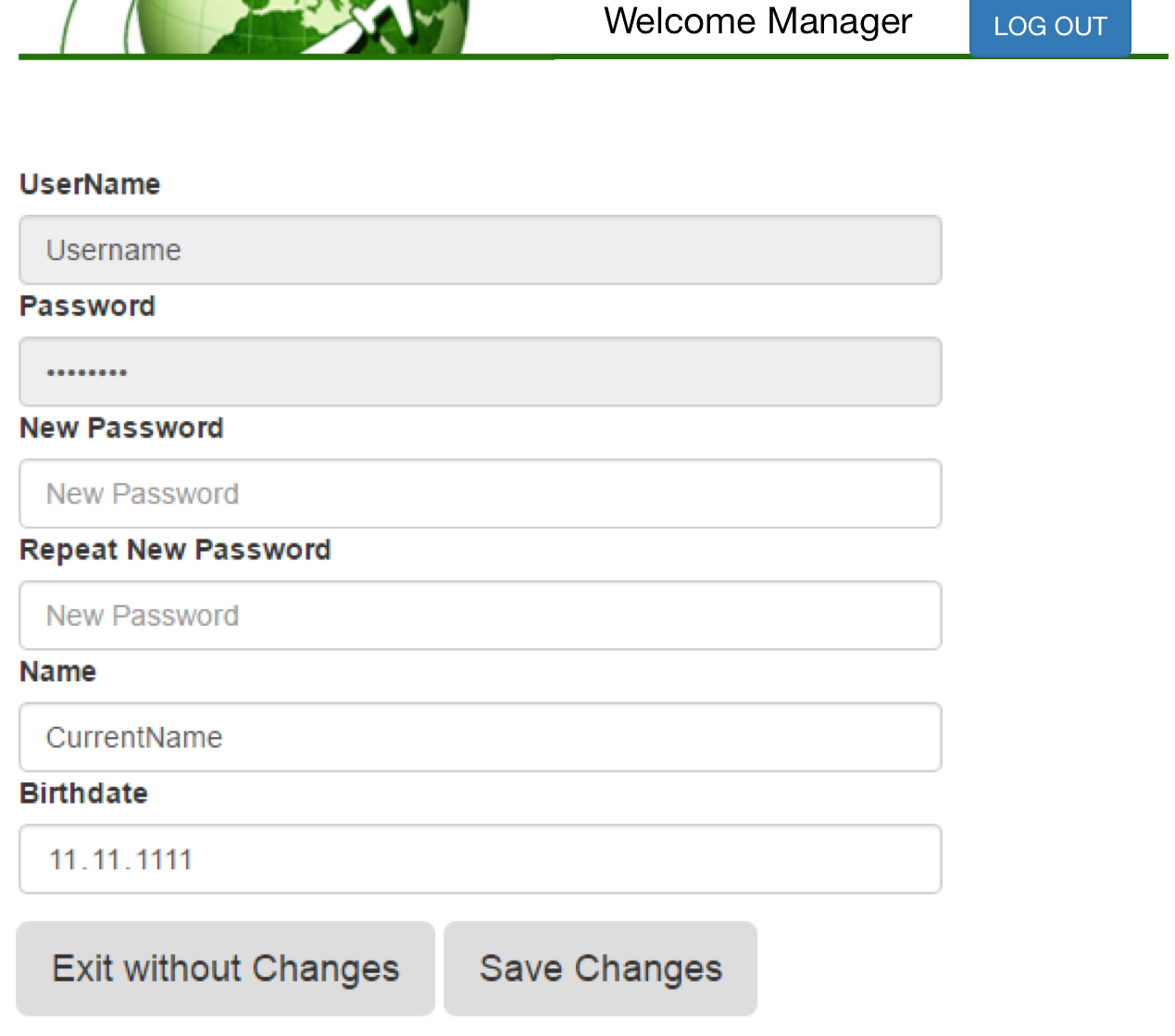
FROM reservation\_authority

WHERE user\_name = @username AND

password = @password AND

user\_name IN (SELECT user\_name FROM reservation\_authority NATURAL JOIN manager)

### Manager Manage Account Screen



**Inputs:** @username, @newusername, @password, @passwordagain, @name, @birthdate

**Process:** In this page that is provided for managers, managers are able to change their passwords, change the possible mistakes in their names, or in their birthdates. After updating the fields manager clicks on ‘Save Changes’ button to update his/her account.

**SQL Statements:**

**View Manager Details**

SELECT \*

FROM reservation\_authority

WHERE user\_name = @username

**Change Username**

UPDATE reservation\_authority

SET user\_name = @newusername

WHERE user\_name = @username

**Change Password**

UPDATE reservation\_authority

SET password = @password

WHERE user\_name = @username AND  
 @password = @passwordAgain

**Change Name**

UPDATE staff

SET name = @name

WHERE user\_name = @username

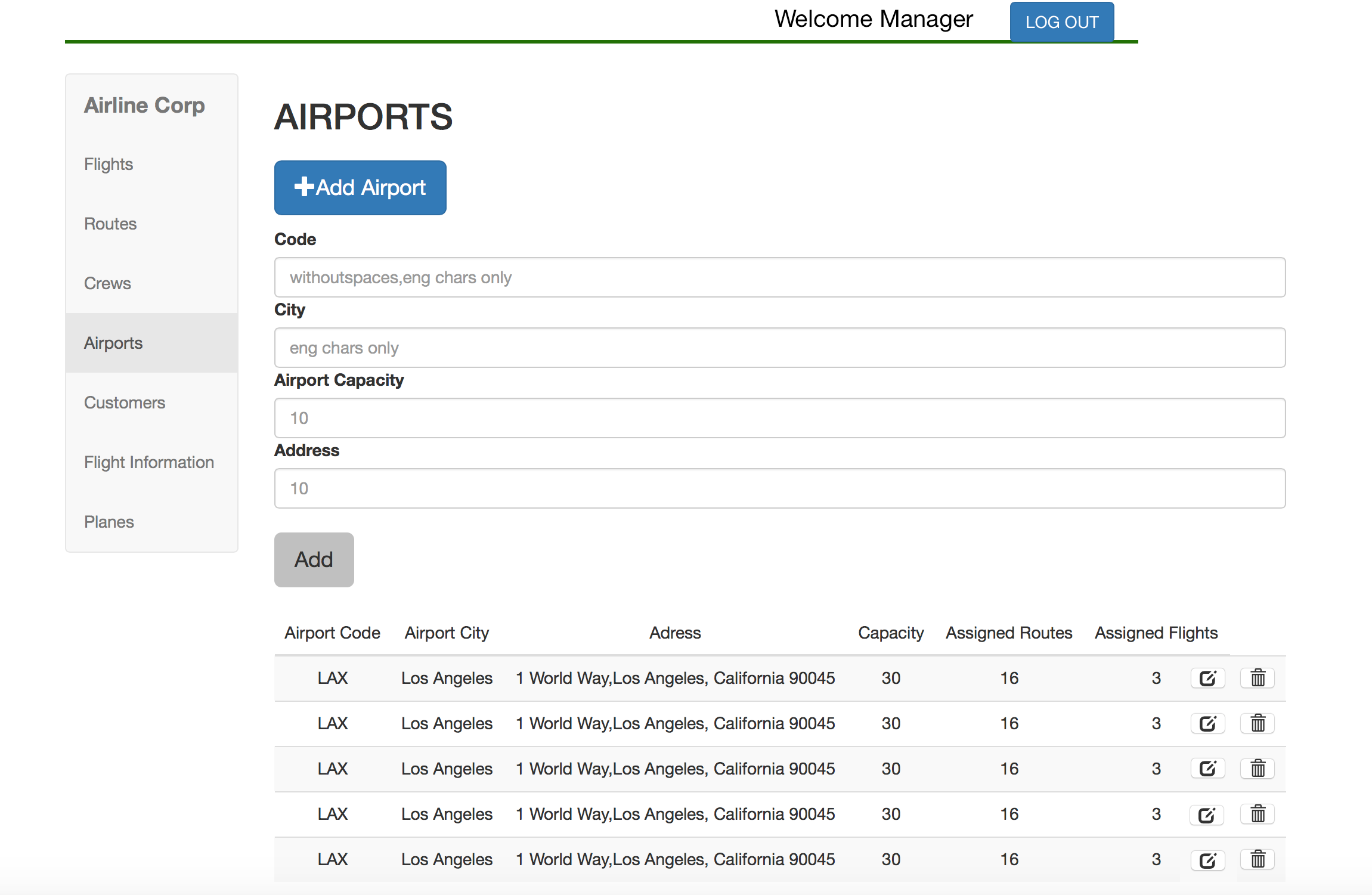
**Change Birthdate**

UPDATE staff

SET birthdate = @birthdate

WHERE user\_name = @username

### Manage Airports Screen



**Inputs:** @airport\_id, @airport\_capacity, @address, @city\_name

**Process:** The manager clicks on ‘Airports’ tab in order to view all airports registered to the system. The manager clicks on ‘Add Airport’ button and the airport fields appear on the screen. He/she enters code(id), city, capacity, address attributes and clicks ‘Add’ button to add a new airport. In order to edit the attributes of the airport the manager clicks on the edit symbol next to each entry. Similarly, the manager clicks delete symbol to delete the airport from the system.

**SQL Statements:**

**View Airports**

SELECT \*

FROM airport

**Add a New Airport**

INSERT INTO airport

VALUES(@airport\_id, @airport\_capacity, @address, @city\_name)

**Delete Airport**

DELETE FROM airport

WHERE airport\_id = @airport\_id

**Update Airport Capacity**

UPDATE airport

SET airport\_capacity = @airport\_capacity

WHERE airport\_id = @airport\_id

**Update Airport Address**

UPDATE airport

SET address = @address

WHERE airport\_id = @airport\_id

**Update Airport City**

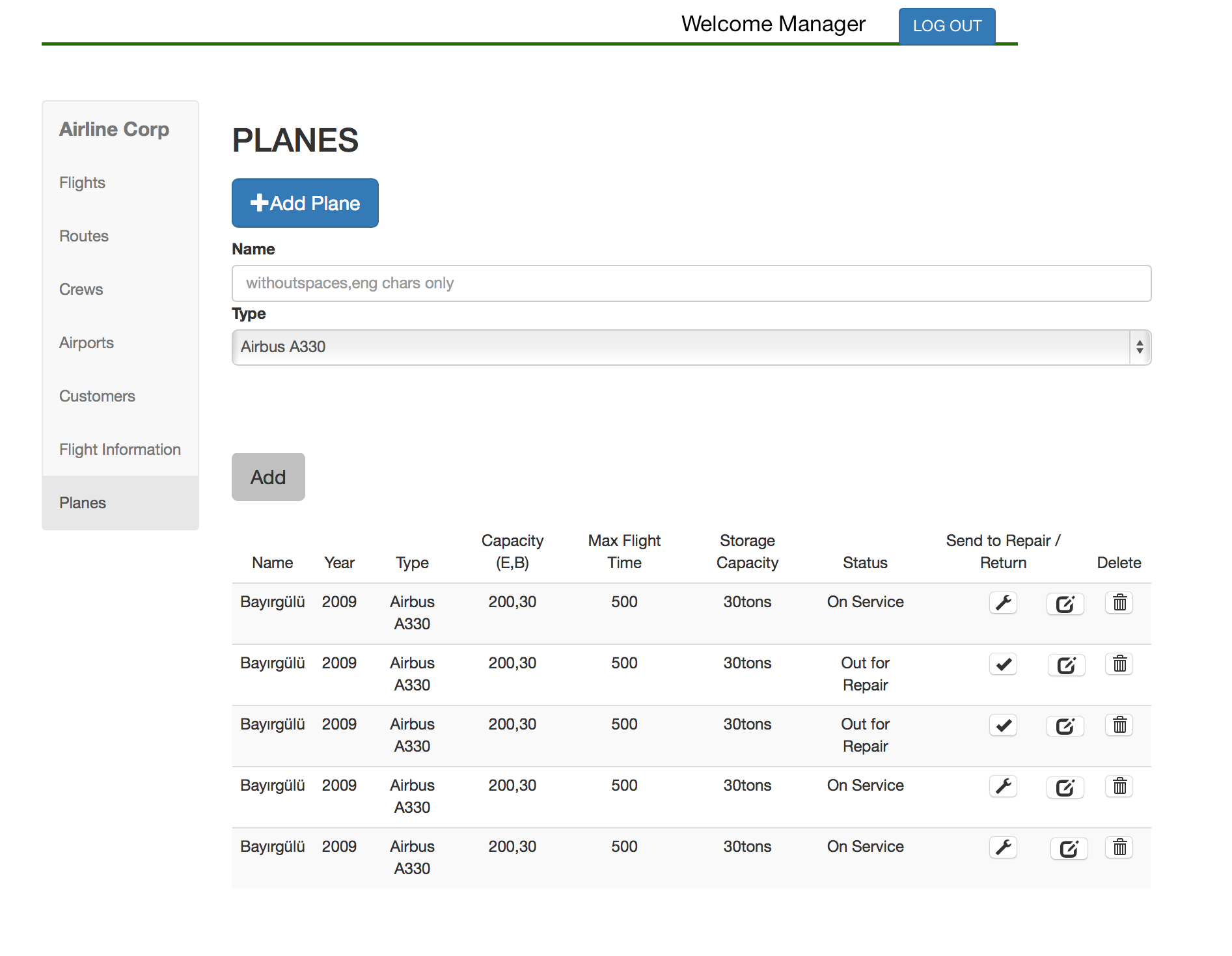
UPDATE airport

SET city\_name = @city\_name

WHERE airport\_id = @airport\_id

Report 6.3.5. ‘Total Number of Tickets and Reservations Associated with Each Flight and the List of All Tickets and Reservations of the Flight’ is used in the screen.

### Manage Airports Screen



**Inputs:** @plane\_name, @plane\_type\_id, @available, @max\_flight\_time, @capacity, @storage\_capacity

**Process:** The manager clicks on ‘Planes’ tab in order to view all planes registered to the system. The manager clicks on ‘Add Plane’ button and the plane fields appear on the screen. He/she enters plane name, production year, and selects plane\_type\_id attributes and clicks ‘Add’ button to add a new plane. In order to edit the attributes of the plane the manager clicks on the edit symbol next to each entry. When edit is clicked the system enables manager to change the attributes just like in airport schema. Similarly, the manager clicks delete symbol to delete the plane from the system.

**SQL Statements:**

**List All Planes**

SELECT plane\_name, production\_year, plane\_type\_id, capacity, max\_flight\_time, storage\_capacity, available

FROM plane

**Add New Plane**

INSERT INTO plane

VALUES(@plane\_name, 1, @plane\_type\_id)

**Delete Plane**

DELETE FROM plane

WHERE plane\_name = @plane\_name

**Update Plane Type**

UPDATE plane

SET plane\_type\_id = @plane\_type\_id

WHERE plane\_name = @plane\_name

**Update Plane Capacity**

UPDATE plane\_type

SET capacity = @capacity

WHERE plane\_type\_id = @plane\_type\_id AND

plane\_name = @plane\_name

**Update Plane Maximum Flight Time**

UPDATE plane\_type

SET max\_flight\_time = @max\_flight\_time

WHERE plane\_type\_id = @plane\_type\_id AND

plane\_name = @plane\_name

**Update Plane Storage Capacity**

UPDATE plane\_type

SET storage\_capacity = @storage\_capacity

WHERE plane\_type\_id = @plane\_type\_id AND

plane\_name = @plane\_name

**Send Plane to Repair**

UPDATE plane

SET available = 0

WHERE plane\_type\_id = @plane\_type\_id AND

plane\_name = @plane\_name

**Mark Plane as Available After Repair**

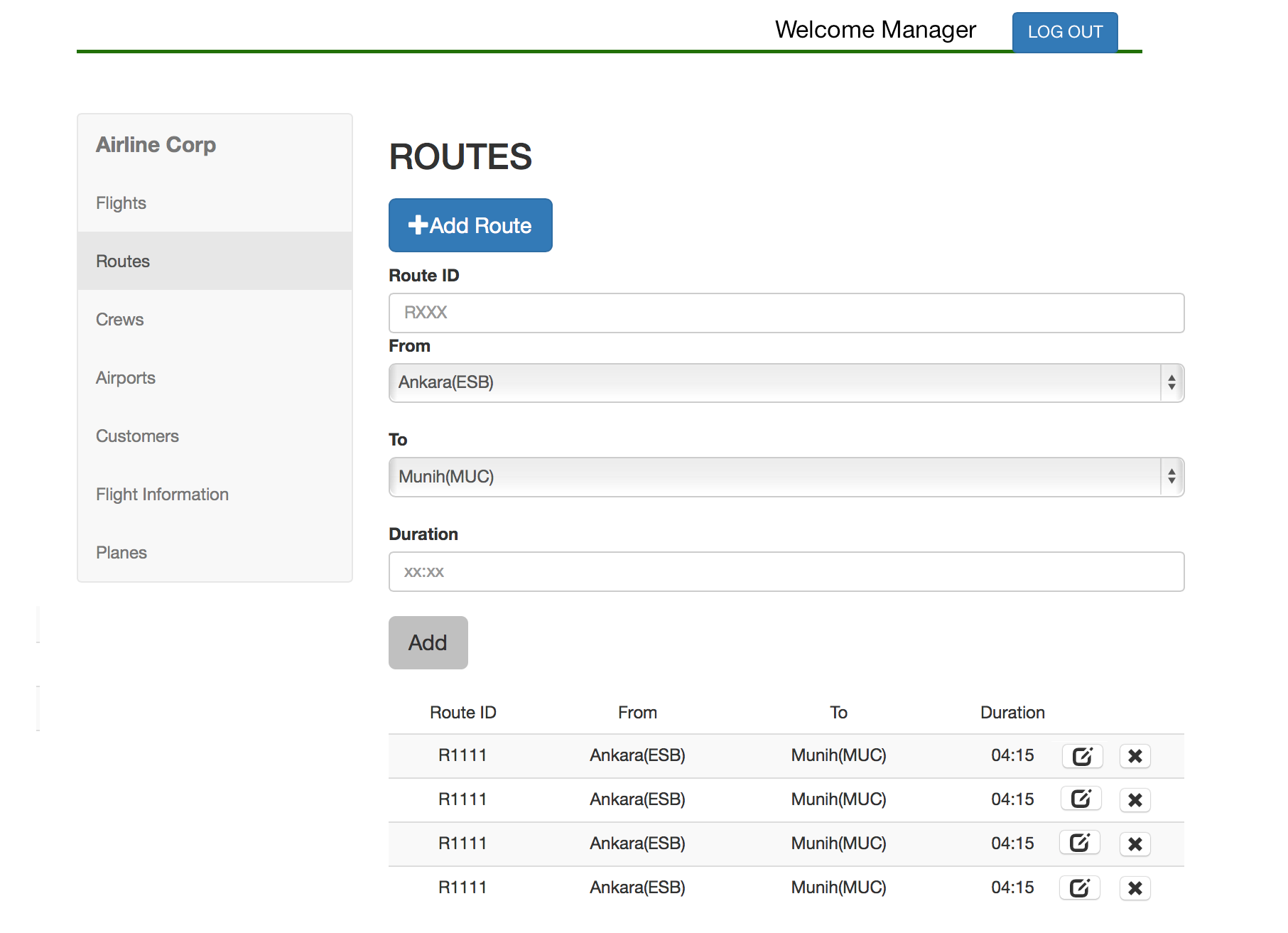
UPDATE plane

SET available = 1

WHERE plane\_type\_id = @plane\_type\_id AND

plane\_name = @plane\_name

### Manage Routes Screen



**Inputs:** @route\_id, @flight\_duration, @departs, @arrives

**Process:** The manager clicks on ‘Routes’ tab in order to view all routes registered to the system. The manager clicks on ‘Add Route’ button and the plane fields appear on the screen. He/she enters route id, arrival and departure airports, and duration attributes and clicks ‘Add’ button to add a new route. In order to edit the attributes of the route the manager clicks on the edit symbol next to each entry. When edit is clicked the system enables manager to change the attributes just like how the fields appear to add a route. Similarly, the manager clicks delete symbol to delete the plane from the system.

**SQL Statements:**

**View Routes**

SELECT \*

FROM route

**Add a New Route**

INSERT INTO route

VALUES(@route\_id, @flight\_duration, @departs, @arrives)

**Delete Route**

DELETE FROM route

WHERE route\_id = @route\_id

**Update Route Duration**

UPDATE route

SET flight\_duration = @flight\_duration

WHERE route\_id = @route\_id

**Update Route Departure Airport**

UPDATE route

SET departs = @departs

WHERE route\_id = @route\_id

**Update Route Arrival Airport**

UPDATE route

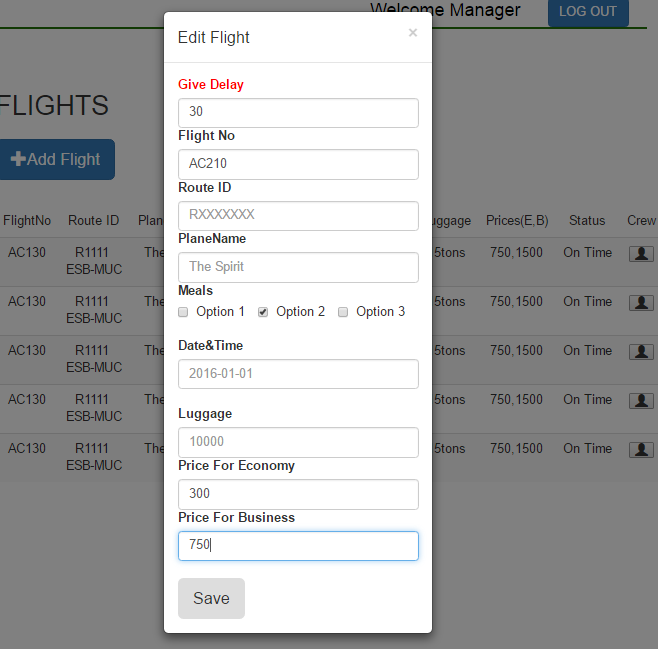
SET arrives = @arrives

WHERE route\_id = @route\_id

### **https://lh3.googleusercontent.com/vUzdqeG8_O-U2ck0CtoZh96hhMZM-e5AWJ8TA-_2ywUiA82MlsvPt5BwbYgA_G7q5R_E4Rc2cpm9UfR4ePCONo7G3OQhhpfuLY-1YHws9fMih3wp9wROz4HIQtsQZvCpf-o9CW2H**Manage Flights Screen

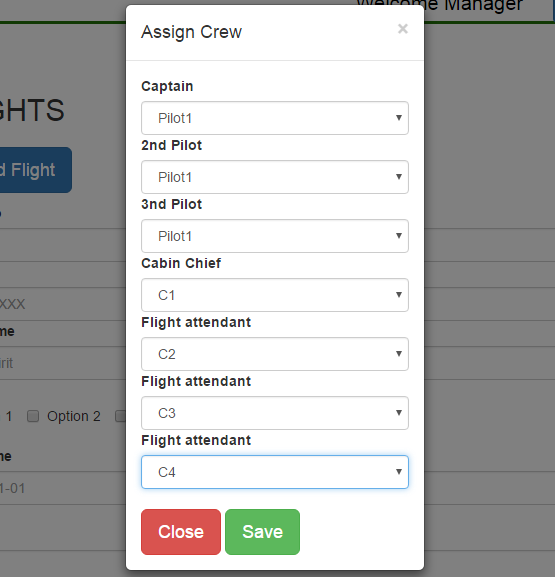
**Inputs:** @plane\_name, @route\_id, @date, @departure\_time, @flight\_id, @meals, @luggage, @price\_bus, @price\_econ, @staff\_id

**Process:** The manager clicks on ‘Flights’ tab in order to view all flights registered to the system. The manager clicks on ‘Add Flight’ button and the flight fields appear on the screen. He/she enters flight number, route id, plane name, meal, date, time, luggage, and price attributes and clicks ‘Add’ button to add a new route. In order to edit the attributes of the route the manager clicks on the edit symbol next to each entry. Similarly, the manager clicks delete symbol to delete the flight from the system. When edit is clicked the system enables manager to change the attributes with the ‘Edit Flight’ box shown in the picture:



Edit Flight box appears when the edit icon is clicked.

Additionally, the delay amount can be set from the edit screen. The manager clicks crew icon next to each flight to see the Assign Crew box as shown in the below picture. The manager can select pilots and flight attendance to assign to the flight and click ‘Save’:

****

Assign Crew box appears when the crew icon is clicked.

**SQL Statements:**

**List All Flights**

SELECT \*

FROM flight

**Add a New Flight**

INSERT INTO flight

VALUES(@plane\_name, @route\_id, @flight\_id, @date, @departure\_time, 0, @meals, @luggage, @price\_bus, @price\_econ)

**Delete Flight**

DELETE FROM flight

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Id**

UPDATE flight

SET flight\_id = @flight\_id

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Business Price**

UPDATE flight

SET business\_price= @price\_bus

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Economy Price**

UPDATE flight

SET economy\_price= @price\_econ

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Meals**

UPDATE flight

SET meals = @meals

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Luggage**

UPDATE flight

SET luggage = @luggage

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Date**

UPDATE flight

SET date = @date

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Time**

UPDATE flight

SET departure\_time = @time

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Plane**

UPDATE flight

SET plane\_name= @plane\_name

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Delay**

UPDATE flight

SET delay = @delay

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Update Flight Luggage**

UPDATE flight

SET delay = @delay

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

**Assign Crew to Flight**

INSERT INTO flight\_crew

SELECT C.staff\_id, F.plane\_name, F.route\_id, F.date, F.departure\_time

FROM crew C, flight F, route R

WHERE C.staff\_id = @staff\_id AND

C.current\_location = R.departs AND

R.route\_id = @route\_id AND

F.plane\_name = @plane\_name AND

F.route\_id = @route\_id

**Delete Crew from Flight**

DELETE FROM flight\_crew

WHERE staff\_id = @staff\_id AND

plane\_name = @plane\_name AND

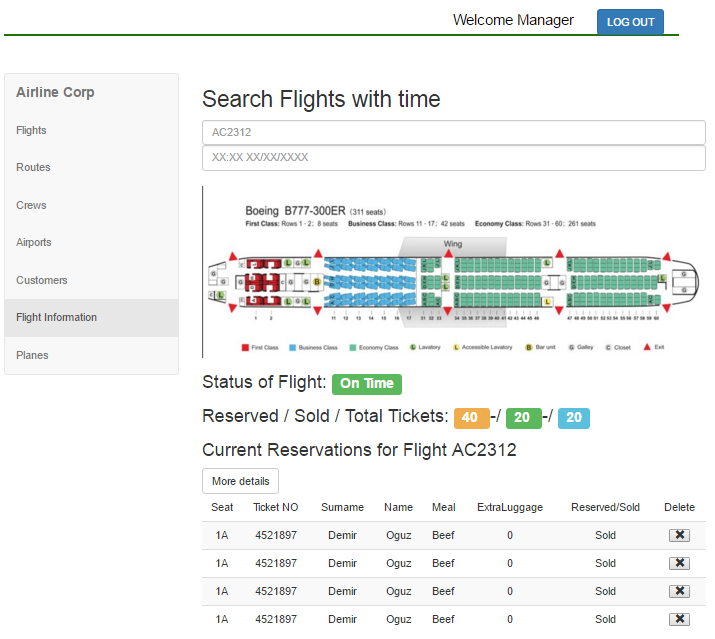
route\_id = @route\_id AND

date = @date AND

departure\_time = @departure\_time

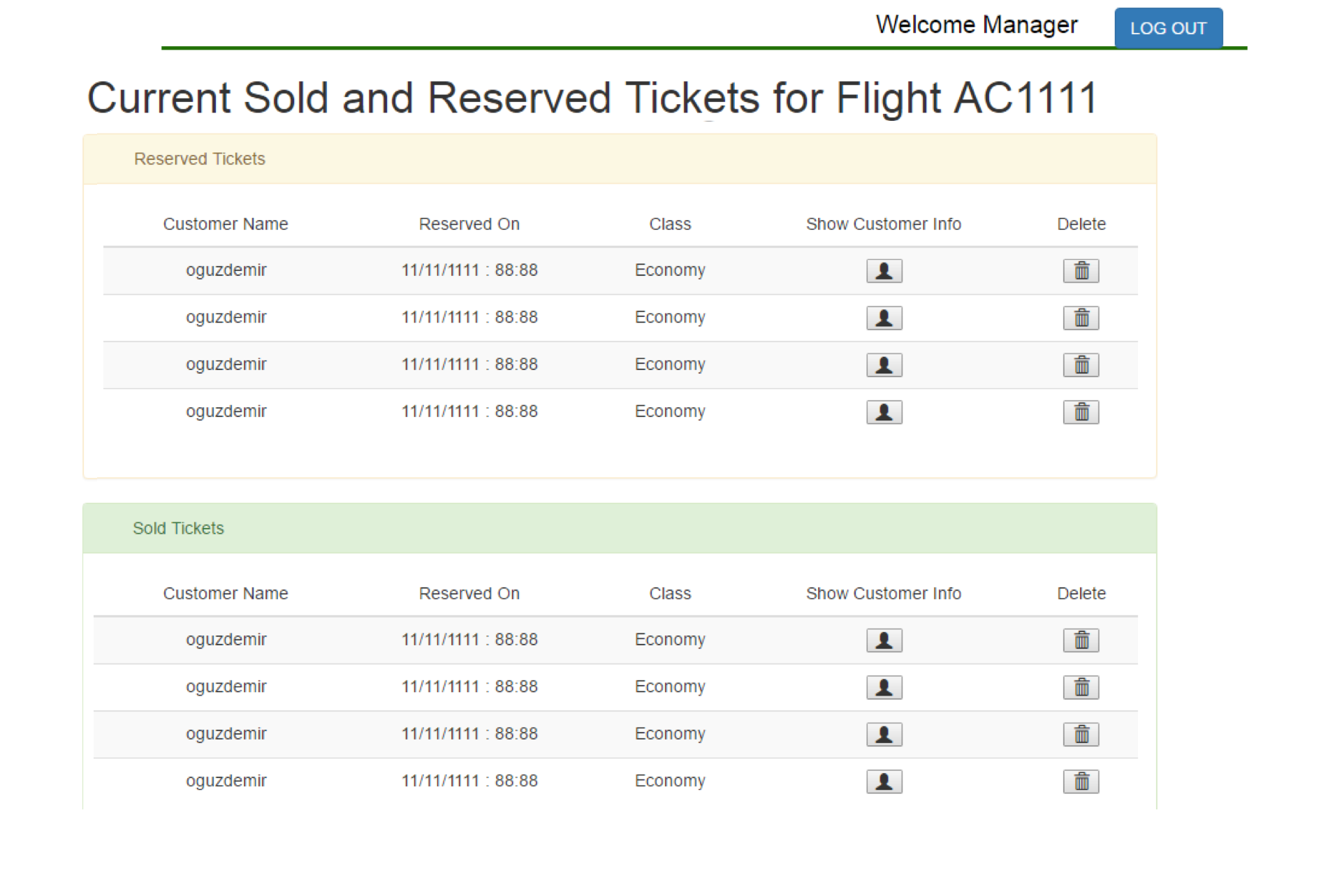
Report 6.3.5. Total Number of Assigned Routes and Flights to Each Airport is used in this screen.

### View Flight Status Screen

****

**Inputs:** @username (from previous pages)

**Process:** The manager clicks ‘Flight Information’ tab to view all flights along with the status, reserved and sold ticket numbers and the list of tickets. The manager clicks on ‘Show Customer Info’ icon to view the customer details. He/she clicks ‘Delete icon to delete reservation or ticket’. When the manager clicks ‘More Details’ button the second screen is displayed where the reservation and ticket information is shown in detail:

The above screen is shown when ‘More Details’ is clicked.

**SQL Statements:**

**List All Flight Details**

SELECT \*

FROM CustomerManagerFlightDetailsView

**List All Reservations&Tickets**

(SELECT user\_name, date, departure\_time, class

FROM reservation

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id

date = @date

departure\_time = @departure\_time) UNION

(SELECT user\_name, date, departure\_time, class

FROM ticket

WHERE plane\_name = @plane\_name AND

route\_id = @route\_id

date = @date

departure\_time = @departure\_time)

**Show Customer Information**

SELECT \*

FROM ManagerCustomerView

**Delete Reservation**

DELETE FROM reservation

WHERE reservation\_no = @reservationno AND  
 user\_name = @username

**Delete Ticket**

DELETE FROM ticket

WHERE ticket\_no = @ticketno AND  
 user\_name = @username

Executes 6.4.2. Mile Sum Trigger After Refunding Ticket

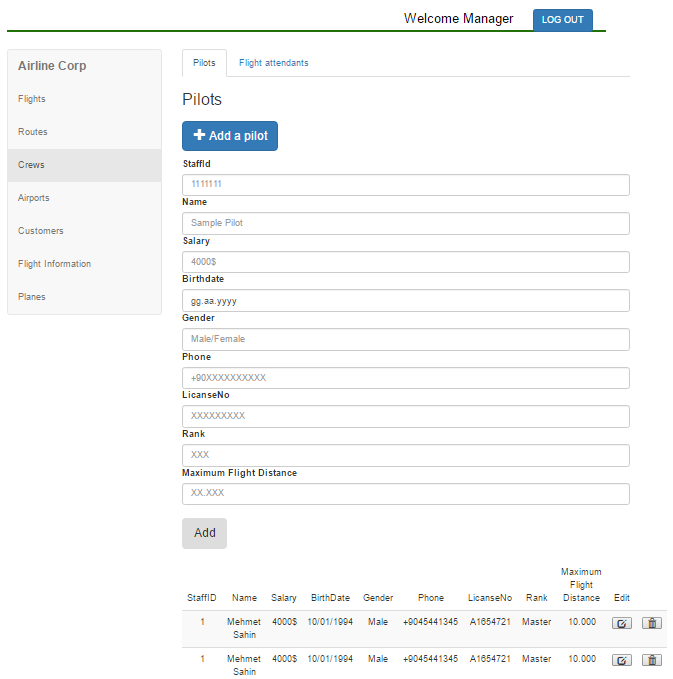
Executes 6.4.3. Cancel Reservation Trigger After Refunding Ticket

Executes 6.4.5. Total Money Trigger After Refunding Ticket

Executes 6.4.6. Pay Penalty Trigger After Refunding Ticket

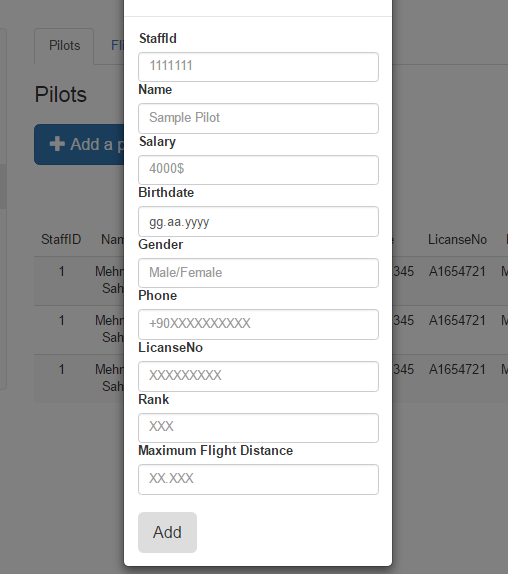
### Manage Staff Screens

#### Manage Pilot Screen

****

**Inputs:** @staff\_id, @max\_flight\_distance, @name, @salary, @birthdate, @gender, @phone, @license\_no, @rank

**Process:** When managers click “Pilots” available at the top of the page, they see the page above. They can add pilots to the system, with the blue button under the “Pilots” label, or they can delete pilots with the trash bin sign available in the same row as the pilot. The image above shows the situation of the page after clicking the “Add a pilot” button. After filling in the necessary blanks provided, the pilot will be added when they click the gray “Add” button. They can also edit pilots’ information with the button which has a square sign. Managers see the pop-up window to edit them shown below:



With filling the required blanks, they can edit the pilot and save it. After saving the choices by clicking the button “Add”, they can return back to Pilots page.

**SQL Statements:**

**View All Pilots**

SELECT \*

FROM pilot

**Add a New Pİlot**

INSERT INTO pilot

VALUES(@staff\_id, @max\_flight\_distance)

INSERT INTO crew

VALUES(@staff\_id, @license\_no, @rank, NULL)

INSERT INTO staff

VALUES (@staff\_id, @name, @salary, @birthdate, getYear() - @birthdate, @gender )

INSERT INTO staff\_phones VALUES(@staff\_id, @phone)

**Update Flight Distance of Pilot**

UPDATE pilot

SET max\_flight\_distance = @max\_flight\_distance

WHEN staff\_id = @staff\_id

**Update Rank of Pilot**

UPDATE crew

SET rank = @rank

WHEN staff\_id = @staff\_id

**Note:** Update current\_location and license\_no queries are the same.

**Update Name of Pilot**

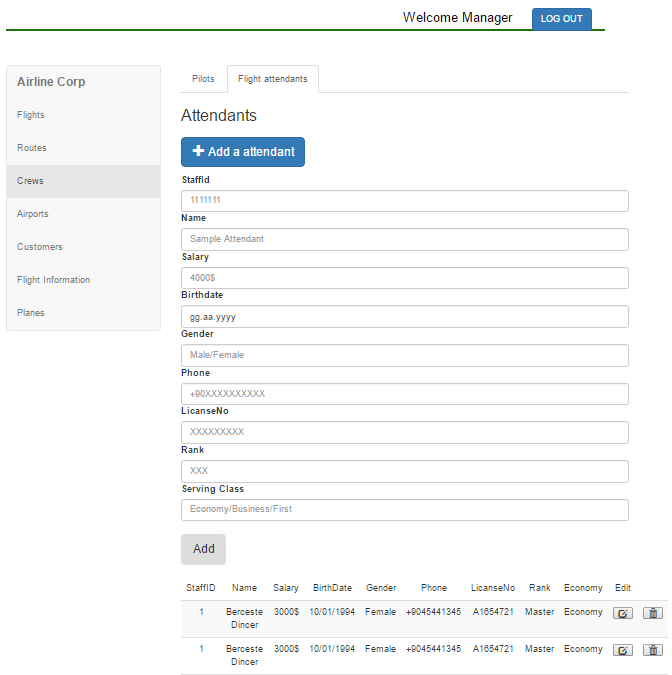
UPDATE staff

SET name = @name

WHEN staff\_id = @staff\_id

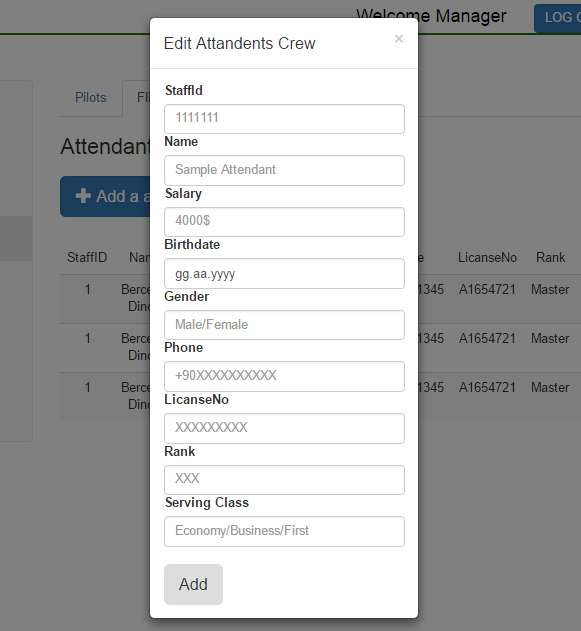
**Note:** Update salary, birthdate, gender queries are the same.

#### Manage Flight Attendant Screen

****

**Inputs:** @staff\_id, @name, @salary, @birthdate, @gender, @phone, @license\_no, @rank, @class\_served

**Process:** When managers click “Flight Attendants” available at the top of the page, they see the page above. They can add flight attendants to the system, with the blue button under the “Attendants” label, or they can delete attendants with the trash bin sign available in the same row as flight. The image above shows the situation of the page after clicking the “Add an attendant” button. After filling in the necessary blanks provided, the attendant will be added when they click the gray “Add” button. They can also edit attendants’ information with the button which has a square sign. Managers see the pop-up window to edit them shown below:

****

With filling the required blanks, they can edit the pilot and save it. After saving the choices by clicking the gray button “Add”, they can return back to Flight Attendants page.

**SQL Statements:**

**View All Flight Attendants**

SELECT \*

FROM flight\_attendance

**Add a New Flight Attendant**

INSERT INTO flight\_attendance

VALUES(@staff\_id, @class\_served)

INSERT INTO crew

VALUES(@staff\_id, @license\_no, @rank, NULL)

INSERT INTO staff

VALUES (@staff\_id, @name, @salary, @birthdate, getYear() - @birthdate, @gender )

INSERT INTO staff\_phones VALUES(@staff\_id, @phone)

**Update Class Served of Flight Attendant**

UPDATE flight\_attendance

SET flight\_class\_served = @class\_served

WHEN staff\_id = @staff\_id

**Update Rank of Flight Attendant**

UPDATE crew

SET rank = @rank

WHEN staff\_id = @staff\_id

**Note:** Update current\_location and license\_no queries are the same.

**Update Name of Flight Attendant**

UPDATE staff

SET name = @name

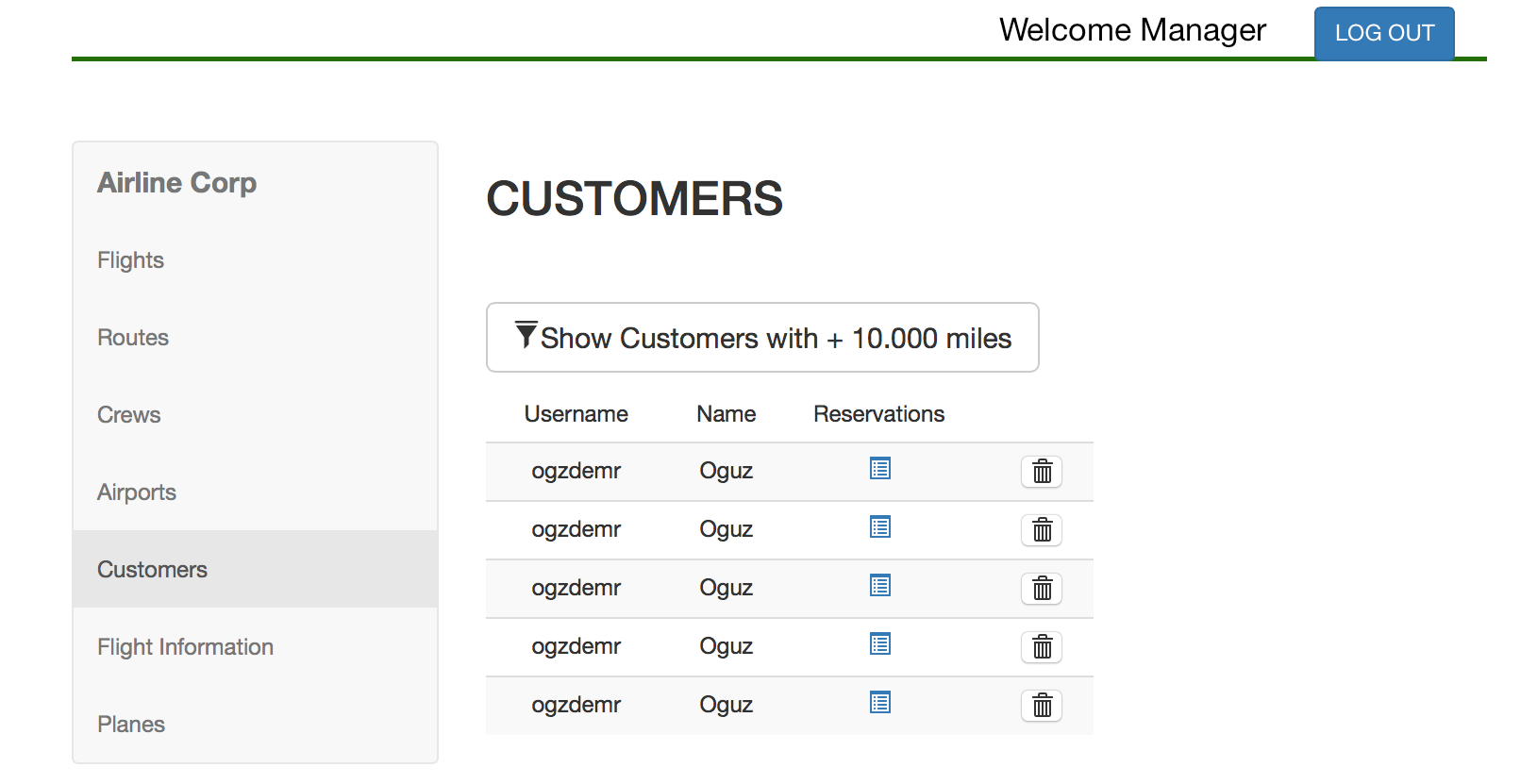
WHEN staff\_id = @staff\_id

**Note:** Update salary, birthdate, gender queries are the same.

#### Manage Salesperson/Ticketing Agent Screens

The screens for managing salesperson and ticketing agent are the same as the pilot and flight attendant screens. Only the specific attributes are changed.

### View Customers Screen



**Inputs:** @username (from previous pages), @customername

**Process:** The manager clicks ‘Customers’ tab to view all customers. He/she select ‘Show Customers with +10.000 miles’ to view customers who earned promotions. When the delete icon is clicked the customer is deleted from the system. When reservation is clicked the below screen is shown:



The manager can view all reservations and tickets of a specific customer from this screen.

**SQL Statements:**

**View All Customers**

SELECT \*

FROM ManagerCustomerView

**View Tickets and Reservations Of Customer**

(SELECT date, time, departs, arrives, flight\_duration, class

FROM CustomerReservationView) UNION

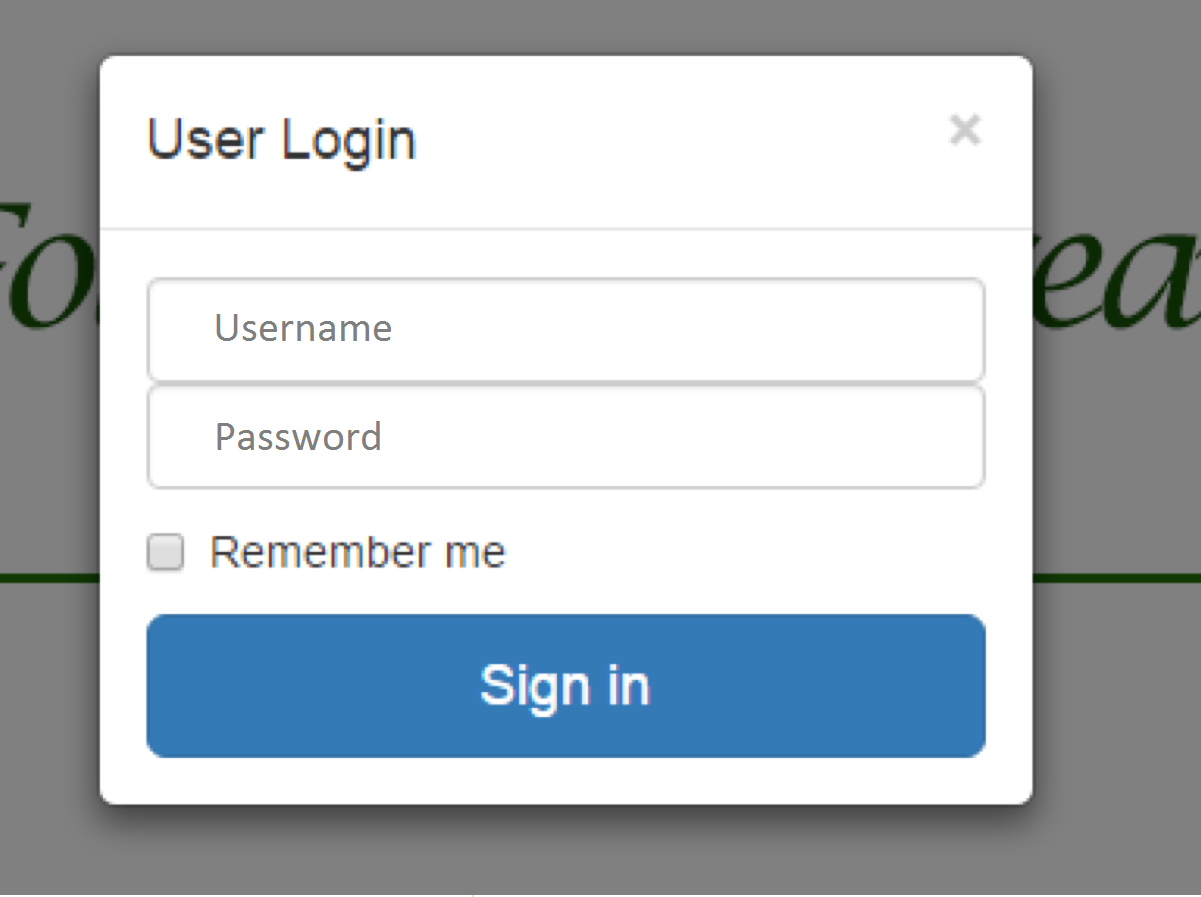
(SELECT date, time, departs, arrives, flight\_duration, extra\_luggage, class

FROM CustomerTicketView)

## Salesperson User Interface Design

The screens salesperson can access are described in detail.

### Salesperson Login Screen



**Inputs:** @username, @password

**Process:** The manager clicks ‘Login’ button. He/she enters his/her username and password in order to access to manager operations.

**SQL Statements:**

**Login to the System**

SELECT \*

FROM reservation\_authority

WHERE user\_name = @username AND

password = @password AND

user\_name IN (SELECT user\_name FROM reservation\_authority NATURAL JOIN salesperson)

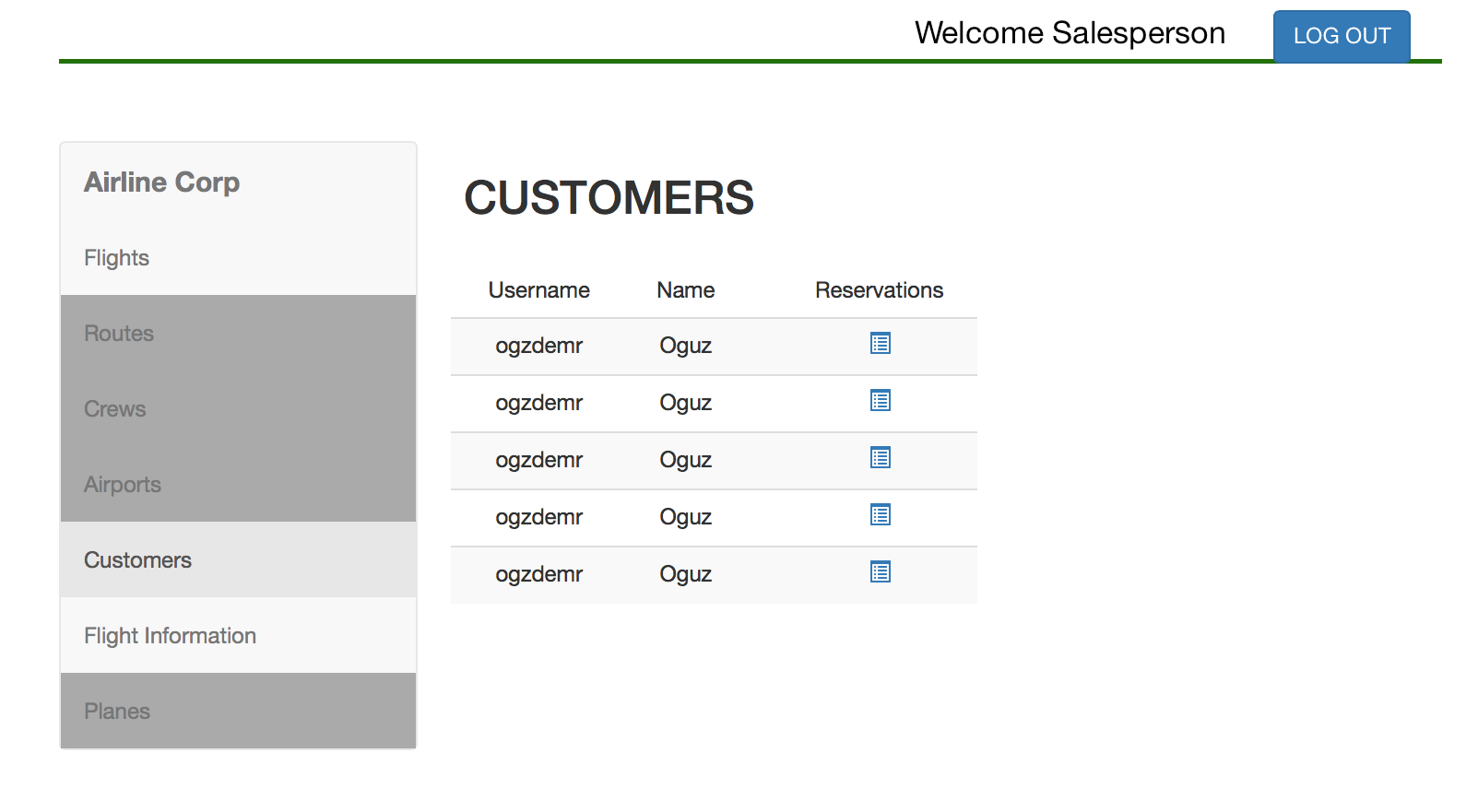
### Salesperson Manage Account Screen

Since both manager and salesperson are reservation authoritative, the changes on account are reflected to staff and reservation\_authoritative tables. Hence, the user interface design and the queries for salesperson manage account screen is exactly the same as the manager’s. The user interface and queries can be found in section 5.2.2. of the report.

### Salesperson View Flight Status Screen

Just like the manager, the salesperson can view all flights registered to the system and view the tickets and reservations for the flights. The user interface and queries for the salesperson view flight status is the same as the manager’s flight details screen. The salesperson can also use this screen in order to inform the customer about the available flights, i.e. the salesperson can use this page to search for available flights. The user interface and queries can be found in section 5.2.7. of the report.

### Salesperson Customer View



**Inputs:** @username (from previous pages), @customername

**Process:** The salesperson clicks ‘Customers’ tab to view all customers. The username and name of the customers are shown to the salesperson. Then salesperson clicks to ‘Reservations’ icon next to each customer in order to view all reservations and tickets of the selected customer. (Reservations&Tickets page explain in 5.3.5. )

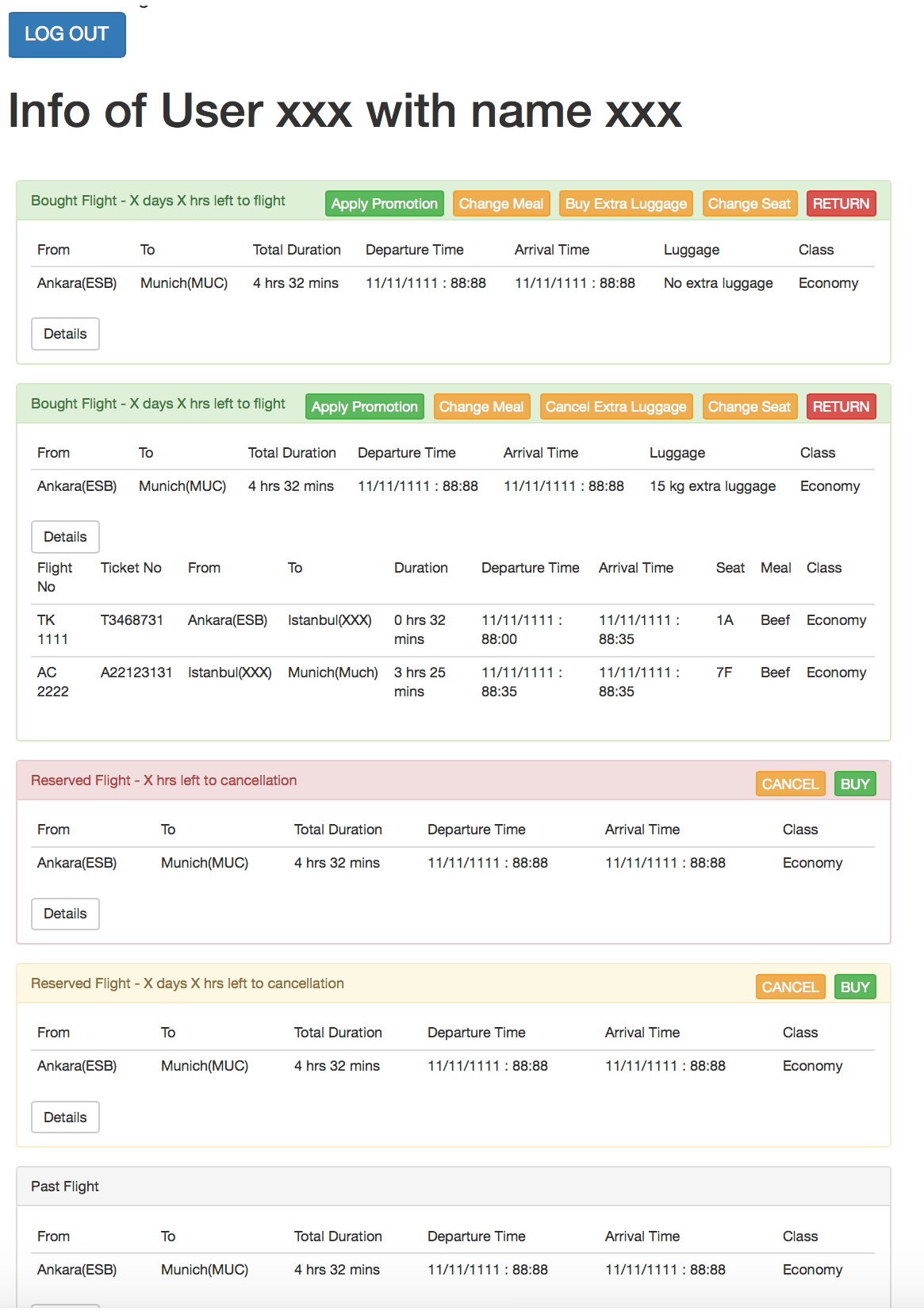
**SQL Statements:**

**View All Customers**

SELECT \*

FROM ManagerCustomerView

### Salesperson Customer Reservations &Tickets View



**Inputs:** @curtomername (passed from salesperson customer view screen), @reservationno, @ticketno, @selectedPromotion, @extraluggage, @meal, @class, @seat\_no

When salesperson selects reservations from the customer screen the page above is shown. The salesperson can see the existing reservations of the selected customer in detail. When he/she click “Details” button, ticket’s every single available information is provided. The salesperson is able to perform any operation the customer can perform in this page in order to help the customer with the process.

The salesperson clicks ‘Cancel’ button to cancel the reservation and clicks ‘Buy’ button to purchase the ticket. He/she clicks ‘Apply Promotion’ button to select a campaign or sale to apply to the ticket from the list of available promotions. Salesperson clicks ‘Buy Meal’ to select a meal and ‘Change Meal’ to update the meal choice. He/she clicks ‘Buy Extra Luggage’ to specify the amount of extra luggage and clicks ‘Cancel Extra Luggage’ to delete the existing luggage. The customer clicks ‘Choose Seat’ to select a seat from the list of seats and clicks ‘Change Seat’ to update selection. When customer clicks ‘Return’ button the ticket is refunded. He/she selects ‘Return’ button to refund the ticket. This page allows salesperson to handle reservation & purchasing operations on behalf of the customers.

**SQL Statements:**

**View Tickets and Reservations Of Customer**

(SELECT date, time, departs, arrives, flight\_duration, class

FROM CustomerReservationView) UNION

(SELECT date, time, departs, arrives, flight\_duration, extra\_luggage, class

FROM CustomerTicketView)

**View all Tickets**

SELECT \*

FROM CustomerTicketView

**Cancel Reservation**

UPDATE reservation

SET cancelled = 1

WHERE reservation\_no = @reservationno AND  
 user\_name = @customername

**Return and Refund Ticket**

DELETE FROM ticket

WHERE ticket\_no = @ticketno AND  
 user\_name = @customername

Executes 6.4.2. Mile Sum Trigger After Refunding Ticket

Executes 6.4.3. Cancel Reservation Trigger After Refunding Ticket

Executes 6.4.5. Total Money Trigger After Refunding Ticket

Executes 6.4.6. Pay Penalty Trigger After Refunding Ticket

**List Promotions**

SELECT \*

FROM promotion

WHERE user\_name = @customername

**Apply Promotion**

UPDATE ticket

SET price = CASE

WHEN @selectedPromotion in (SELECT promotion\_id FROM sale WHERE user\_name = @username) THEN price \* sale\_amount

WHEN @selectedPromotion in (SELECT promotion\_id FROM campaign WHERE user\_name = @username) THEN 0

ELSE price

END

**Buy/Change Meal**

UPDATE ticket

SET meal = @meal

WHERE ticket\_no = @ticketno AND  
 user\_name = @customername

Executes 6.4.7. Total Money Trigger After Buying Meal

**Choose/Change Seat**

UPDATE ticket

SET seat\_no = @seatno

WHERE ticket\_no = @ticketno AND  
 user\_name = @customername

**Register Extra Luggage**

UPDATE ticket

SET extra\_luggage = @extraluggage

WHERE ticket\_no = @ticketno AND  
 user\_name = @customername

**Cancel Extra Luggage**

UPDATE ticket

SET extra\_luggage = 0

WHERE ticket\_no = @ticketno AND  
 user\_name = @customername

**Purchase Ticket**

INSERT INTO ticket

FROM flight NATURAL JOIN route

SELECT (user\_name, plane\_name, route\_id, date, departure\_time, INDEX(ticket\_no), 0, NULL, @class, 0, NULL, CASE WHEN @class = ‘Business’ business\_price ELSE economy\_price END)

WHERE user\_name = @customername AND

route\_id in (SELECT route\_id FROM route where departs = @from and arrives = @to) AND

date = @departuredate AND

time = @departuretime

Executes 6.4.1. Mile Sum Trigger After Purchasing Ticket

Executes 6.4.4. Total Money Trigger After Purchasing Ticket

# ADVANCED DATABASE COMPONENTS

## Views

### Customer Flight View

The customer can only see the flight id, flight time, date, and cities. The plane and assigned crew details should not be visible to the customer.

CREATE VIEW CustomerFlightView(flight\_id, date, time, departs, arrives, price) AS

(SELECT flight\_id, date, departure\_time, departs, arrives, price

FROM flight NATURAL JOIN route )

### Customer Reservation View

The customer can only see the reservation details such as flight id, flight time, date, cities, class and cancelled. The plane and assigned crew details should not be visible to the customer. Moreover, customer can only see his/her own reservations.

CREATE VIEW CustomerFlightView(flight\_id, date, time, departs, arrives, reservation\_no, class, cancelled) AS

(SELECT flight\_id, date, departure\_time, departs, arrives, reservation\_no, class, cancelled

FROM reservation NATURAL JOIN flight NATURAL JOIN route

WHERE user\_name = @userName)

### Customer Flight Details View

The customer can only see details of the flight he/she has reservation or ticket. Moreover, from the plane properties he/she can only see the seat map and cannot access to other plane details such as capacity or max\_flight\_distance.

CREATE VIEW CustomerFlightDetailsView(flight\_id, date, time, departs, arrives, luggage, meal, seat\_map) AS

(SELECT flight\_id, date, time, departs, arrives, luggage, meal, seat\_map

FROM flight NATURAL JOIN route NATURAL JOIN plane NATURAL JOIN plane\_type

WHERE @userName in ((SELECT user\_name

FROM reservation R

WHERE R.flight\_id = flight\_id) union)

(SELECT user\_name

FROM ticket T

WHERE T.flight\_id = flight\_id) union) )

and user\_name = @userName

### Customer Ticket View

The customer can only see the ticket details such as flight id, flight time, date, cities, luggage, seat\_no etc. The plane, penalty amount, assigned crew details should not be visible to the customer. Moreover, customer can only see his/her own tickets.

CREATE VIEW CustomerTicketView (flight\_id, date, time, departs, arrives, ticket\_no, extra\_luggage, meal, class, seat\_no, price) AS

(SELECT flight\_id, date, time, departs, arrives, ticket\_no, extra\_luggage, meal, class, penalty\_amount, seat\_no, price

FROM ticket NATURAL JOIN flight NATURAL JOIN route

WHERE user\_name = @userName)

### Manager Customer View

The manager cannot access to password or passport details of the customers. Other customer information can be accessed by the manager.

CREATE VIEW ManagerCustomerView(user\_name, name, mile\_sum) AS

(SELECT user\_name, name, mile\_sum

FROM customer )

## Stored Procedures

In Airline Company Data Management System stored procedures are used to improve performance, manage consistency, improving security, and increasing robustness. For the procedures that are complex and repetitively executed, we created stored procedures.

### Make Reservation Stored Procedure

When a reservation is made by the customer or salesperson, the reservation is added to the current reservations. Then, the information of reservations available to users of the system is updated. In manager screen the reservation details are also updated. This procedure is repeated whenever a new reservation is made.

### Cancel Reservation Stored Procedure

When a reservation is cancelled by the customer or salesperson, the reservation information is marked as cancelled. Then, the information of reservations available to users of the system is updated. In manager screen the reservation details are also updated. This procedure is repeated whenever a reservation is cancelled.

### Purchase Ticket Stored Procedure

When customer or salesperson purchases the ticket for the flight they made a reservation from, reservation and ticket entities are joined to indicate which reservation is sold. After the details of the ticket are indicated, the meals registered to flight are updated. If extra luggage is specified, the luggage amount for the flight is updated. Then, according to the class of the flight and the seat number the seat map plan of the plane is updated. The manager screen is also updated to include new tickets. This procedure is repeated whenever a ticket is purchased.

### Refund Ticket Stored Procedure

When customer or salesperson cancels an already bought ticket, reservation and ticket entities are joined to indicate which reservation is involved and the reservation is marked as cancelled. Then, the meals registered to flight are updated. If extra luggage was specified, the luggage amount for the flight is updated. Then, according to the class of the flight and the seat number the seat map plan of the plane is updated. The amount of the ticket is refunded to the customer. Moreover, the penalty amount is paid by the customer. The manager screen is also updated to include new tickets. This procedure is repeated whenever a ticket is refunded.

### Delete Airport Procedure

When manager deletes an airport from the system first the operation is postponed until there is a current associated flight in the air. Afterwards, all the routes including the airport as a source or destination airport are deleted from the system. Then, the flights associated with these routes are deleted from the system. The associated planes are marked as available again. Furthermore, all associated reservations are cancelled and the associated customers are notified. If there are sold tickets, the tickets are cancelled and the customers are paid back the ticket amount. Finally, the manager page reservation/ticket details and available flights of the system is updated. This procedure is repeated whenever an airport is deleted from the system.

### Delete Route Procedure

When manager deletes a route from the system first the operation is postponed until there is a current associated flight in the air. Afterwards, the flights associated with these routes are deleted from the system. The associated planes are marked as available again. Furthermore, all associated reservations are cancelled and the associated customers are notified. If there are sold tickets, the tickets are cancelled and the customers are paid back the ticket amount. Finally, the manager page reservation/ticket details and available flights of the system is updated. This procedure is repeated whenever a route is deleted from the system.

### Cancel Flight Procedure

When manager deletes a flight from the system first the operation is postponed until there is a current associated flight in the air. Afterwards, the associated planes are marked as available again. Furthermore, all associated reservations are cancelled and the associated customers are notified. If there are sold tickets, the tickets are cancelled and the customers are paid back the ticket amount. Finally, the manager page reservation/ticket details and available flights of the system is updated. This procedure is repeated whenever a flight is deleted from the system.

### Send Plane to Repair Stored Procedure

When manager sends a plane to the repair, the plane is marked as unavailable. Then, the flights using these planes are cancelled. All associated reservations are cancelled and the associated customers are notified. If there are sold tickets, the tickets are cancelled and the customers are paid back the ticket amount. Finally, the manager page reservation/ticket details and available flights of the system is updated. This procedure is repeated whenever a plane is sent to repair.

## Reports

### Total Number of Customers Registered to the System, Total Number of Customers with Reservation, Total Number of Customers with Tickets

Calculates the number of customers that are registered to the Airline Company Data Management System, the number of customers that have reservation and the number of customers who has purchased ticket.

WITH allReservationsAndTickets( reservationCount, saleCount ) AS

( SELECT COUNT (distint R.user\_name), COUNT(distinct T.user\_name)

FROM reservation R, ticket T)

SELECT COUNT(C.user\_name), reservationCount, saleCount

FROM allReservationsAndTickets, cutomer C

### Total Number of Available Flights, Total Number of Current Reservations, Total Number of Purchased Tickets and The Total Amount of Money Spent by the Customers

Calculates the total number of available flights, the total number of current reservations to these flights, the number of sold tickets from these flights and the total amount of money the customers paid for tickets.

WITH totalMoney( reservationCount, saleCount ) AS

( SELECT COUNT (R.\*), COUNT(distinct T.\*)

FROM reservation R, ticket T)

SELECT COUNT(F.\*), reservationCount, saleCount, SUM(price)

FROM Flight, totalMoney, Ticket

### Total Number of Employees in Each Role and the Average Salary of Each Role

Calculates the total number of employees in each role and the average value of their salaries.

WITH pilots(pilot\_count, pilot\_avg\_sal) AS

( SELECT COUNT(\*), AVG(salary)

FROM Pilot )

flightattendance(fa\_count, fa\_avg\_sal) AS

( SELECT COUNT(\*), AVG(salary)

FROM FlightAttendance)

managers(manager\_count, manager\_avg\_sal) AS

( SELECT COUNT(\*), AVG(salary)

FROM Manager )

salespersons(salesperson\_count, salesperson\_avg\_sal) AS

( SELECT COUNT(\*), AVG(salary)

FROM salesperson)

SELECT unique (pilot\_count, pilot\_avg\_sal, fa\_count, fa\_avg\_sal, manager\_count, manager\_avg\_sal salesperson\_count, salesperson\_avg\_sal)

FROM pilots, flightattendance, managers, salepersons

### Total Number of Tickets and Reservations Associated with Each Flight and the List of All Tickets and Reservations of the Flight

Calculates the total number of tickets and reservations for each flight and lists all tickets and reservations for the flight.

SELECT count(reservation\_no, user\_name), count(ticket\_no, user\_name), reservation.\*, ticket\*

FROM ticket NATURAL JOIN reservation

WHERE route\_id = @route\_id AND

plane\_name = @plane\_name AND

date = @date AND  
 departure\_time = @departuretime

### Total Number of Assigned Routes and Flights to Each Airport

Calculates the total number of assigned routes and flight for each airport.

SELECT count(route\_id), count(route\_id, plane\_name, date, departure\_time)

FROM flight

WHERE route\_id = @route\_id AND

plane\_name = @ plane\_name AND

date = @date AND

departure\_time = @departure\_time

GROUP BY airport\_name

## Triggers

### Mile Sum Trigger After Purchasing Ticket

After a tuple is inserted to ticket, the mile\_sum attribute of the associated customer is increased by the total\_mile amount of the associated flight.

### Mile Sum Trigger After Refunding Ticket

After a ticket is refunded and the tuple is deleted from ticket, the mile\_sum attribute of the associated customer is decreased by the total\_mile amount of the associated flight.

### Cancel Reservation Trigger After Refunding Ticket

After delete operation on ticket, the corresponding row will be updated as cancelled from reservations.

### Total Money Trigger After Purchasing Ticket

After a tuple is inserted to ticket, the total\_money attribute of the associated customer is decreased by the price amount of the associated flight.

### Total Money Trigger After Refunding Ticket

After a ticket is refunded and the tuple is deleted from ticket, the total\_money attribute of the associated customer is increased by the price amount of the associated flight.

### Pay Penalty Trigger After Refunding Ticket

After a ticket is refunded and the tuple is deleted from ticket, the total\_money attribute of the associated customer is decreased by the penalty\_amount of the associated flight.

### Total Money Trigger After Buying Meal

After a new tuple is inserted to the ticket, the constant price of the meal is deduced from the total\_money attribute of customer.

### Total Money Trigger After Updating Meal

After update on the meal attribute of the ticket, the price of the old meal is added to the total\_money attribute and the price of the new meal is deduced form the total\_money of customer.

### Total Money Trigger After Registering Extra Luggage

After a luggage is registered to a ticket, the constant price of the extra luggage is deduced from the total\_money attribute of customer.

### Total Money Trigger After Cancelling Extra Luggage

After a luggage is cancelled a ticket, the constant price of the extra luggage is refunded to the total\_money attribute of customer.

## Constraints

### Minimum Salary Constraint for Staff

The salary of a staff cannot be lower than the minimum wage 1300 TL.

### Capacity Constraint for Planes Assigned to Flights

A plane with more than 500 passenger capacity cannot be assigned to a flight that has total\_mile lower than 3000 miles.

Similarly, a plane with less than 500 passenger capacity cannot be assigned to a flight with total\_mile higher than 3000.

### Simultaneous Flights Constraint for Customer Flights

Ticket cannot have a pair of exact same date and departure\_time values for the same customer.

### Extra Luggage Constraint for Ticket

Ticket cannot have a more than two extra\_luggage.

### Passenger Capacity Constraint for Plane

A plane cannot be assigned to a flight if the total passengers for the plane (number of tickets) are larger than the capacity of the plane.

### Maximum Travel Time Constraint for Plane

A plane cannot be assigned to a flight if the flight\_duration of the flight is larger than the max\_flight\_time of the plane.

### Seat Number Constraint for Flight Class

Seat numbers for the business class cannot be higher than 1/3 of the largest seat number. (Front seats are reserved for business class)

### Location Constraint for Crew Assignment

The crew cannot be assigned to flights if their current\_location is not the same as the departure airport of the flight.

### Pilot Number Constraint for Pilot Assignment

The number of pilots assigned to a flight cannot be less than 2.

### Purchasing Constraint for Customer

The customer can only purchase a ticket, meal or luggage if the total\_money attribute of the customer is larger than or equal to the amount of the payment.

### Total Money Constraint for Customer

The total\_money attribute of customer cannot be below 0.

# IMPLEMENTATION PLAN

To manage the data flow in our project, we would like to use MySQL Server and we will be maintaining it with Java. For our application functionalities and user interface in our management system’s website, we will be using PHP, Bootstrap, HTML, CSS and JavaScript.