



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

1.	REVISED E/R MODEL	5
1.1.	Changes Made in the E/R Diagram	5
1.2.	Revised E/R Diagram.....	7
2.	RELATION SCHEMAS.....	8
2.1.	Staff.....	8
2.2.	Staff Phones.....	8
2.3.	Reservation Authoritative	9
2.4.	Permissions.....	9
2.5.	Manager	10
2.6.	Salesperson.....	10
2.7.	Crew	11
2.8.	Pilot	11
2.9.	Flight Attendance.....	12
2.10.	City	12
2.11.	Airport	13
2.12.	Route	14
2.13.	Plane Type	14
2.14.	Plane.....	15
2.15.	Flight.....	16
2.16.	Flight Crew.....	16
2.17.	Customer	17
2.18.	Reservation.....	18
2.19.	Ticket.....	19
2.20.	Promotion.....	20
2.21.	Campaign.....	20
2.22.	Sale.....	21
3.	FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES	22
4.	FUNCTIONAL COMPONENTS	23
4.1.	Use Cases/Scenarios	23
4.1.1.	Customer Use Cases	24
4.1.2.	Manager Use Cases	27
4.1.3.	Salesperson Use Cases.....	29
4.2.	Algorithms	31
4.2.1.	Price Related Algorithms	31
4.2.2.	Promotion Related Algorithms	31
4.2.3.	Reservation Related Algorithms	32
4.3.	Data Structures	32
5.	USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS	33
5.1.	Customer User Interface Design	33
5.1.1.	Customer Home Screen	33
5.1.2.	Customer Search Flight Screen with One Way Ticket.....	35
5.1.3.	Result of Flight Search Screen	36
5.1.4.	Login Screen	37
5.1.5.	Create Account Screen	38
5.1.6.	Manage Account Screen.....	39
5.1.7.	Result of Flight Search Screen for Logged in Customers	41
5.1.8.	Customer Reservations & Tickets Screen	43

5.2. Manager User Interface Design	48
5.2.1. Login Screen	48
5.2.2. Manager Manage Account Screen.....	49
5.2.3. Manage Airports Screen	51
5.2.4. Manage Airports Screen	53
5.2.5. Manage Routes Screen	56
5.2.6.....	57
5.2.7. Manage Flights Screen.....	58
5.2.8. View Flight Status Screen	65
5.2.9. Manage Staff Screens	68
5.2.10. View Customers Screen	74
5.3. Salesperson User Interface Design	76
5.3.1. Salesperson Login Screen	76
5.3.2. Salesperson Manage Account Screen.....	77
5.3.3. Salesperson View Flight Status Screen	77
5.3.4. Salesperson Customer View	77
5.3.5. Salesperson Customer Reservations &Tickets View	79
6. ADVANCED DATABASE COMPONENTS	84
6.1. Views.....	84
6.1.1. Customer Flight View.....	84
6.1.2. Customer Reservation View	84
6.1.3. Customer Flight Details View.....	84
6.1.4. Customer Ticket View.....	85
6.1.5. Manager Customer View	85
6.2. Stored Procedures	86
6.2.1. Make Reservation Stored Procedure.....	86
6.2.2. Cancel Reservation Stored Procedure	86
6.2.3. Purchase Ticket Stored Procedure.....	86
6.2.4. Refund Ticket Stored Procedure.....	87
6.2.5. Delete Airport Procedure	87
6.2.6. Delete Route Procedure	87
6.2.7. Cancel Flight Procedure.....	88
6.2.8. Send Plane to Repair Stored Procedure	88
6.3. Reports.....	88
6.3.1. Total Number of Customers Registered to the System, Total Number of Customers with Reservation, Total Number of Customers with Tickets	88
6.3.2. 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
6.3.3. Total Number of Employees in Each Role and the Average Salary of Each Role	89
6.3.4. Total Number of Tickets and Reservations Associated with Each Flight and the List of All Tickets and Reservations of the Flight.....	90
6.3.5. Total Number of Assigned Routes and Flights to Each Airport.....	90
6.4. Triggers.....	91
6.4.1. Mile Sum Trigger After Purchasing Ticket.....	91
6.4.2. Mile Sum Trigger After Refunding Ticket.....	91
6.4.3. Cancel Reservation Trigger After Refunding Ticket	91
6.4.4. Total Money Trigger After Purchasing Ticket	91
6.4.5. Total Money Trigger After Refunding Ticket	91
6.4.6. Pay Penalty Trigger After Refunding Ticket	92
6.4.7. Total Money Trigger After Buying Meal	92
6.4.8. Total Money Trigger After Updating Meal.....	92

6.4.9.	Total Money Trigger After Registering Extra Luggage	92
6.4.10.	Total Money Trigger After Cancelling Extra Luggage.....	92
6.5.	Constraints	93
6.5.1.	Minimum Salary Constraint for Staff	93
6.5.2.	Capacity Constraint for Planes Assigned to Flights.....	93
6.5.3.	Simultaneous Flights Constraint for Customer Flights.....	93
6.5.4.	Extra Luggage Constraint for Ticket.....	93
6.5.5.	Passenger Capacity Constraint for Plane	93
6.5.6.	Maximum Travel Time Constraint for Plane	93
6.5.7.	Seat Number Constraint for Flight Class.....	94
6.5.8.	Location Constraint for Crew Assignment	94
6.5.9.	Pilot Number Constraint for Pilot Assignment	94
6.5.10.	Purchasing Constraint for Customer.....	94
6.5.11.	6.5.11. Total Money Constraint for Customer	94
7.	IMPLEMENTATION PLAN	94

1. REVISED E/R MODEL

1.1. 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:

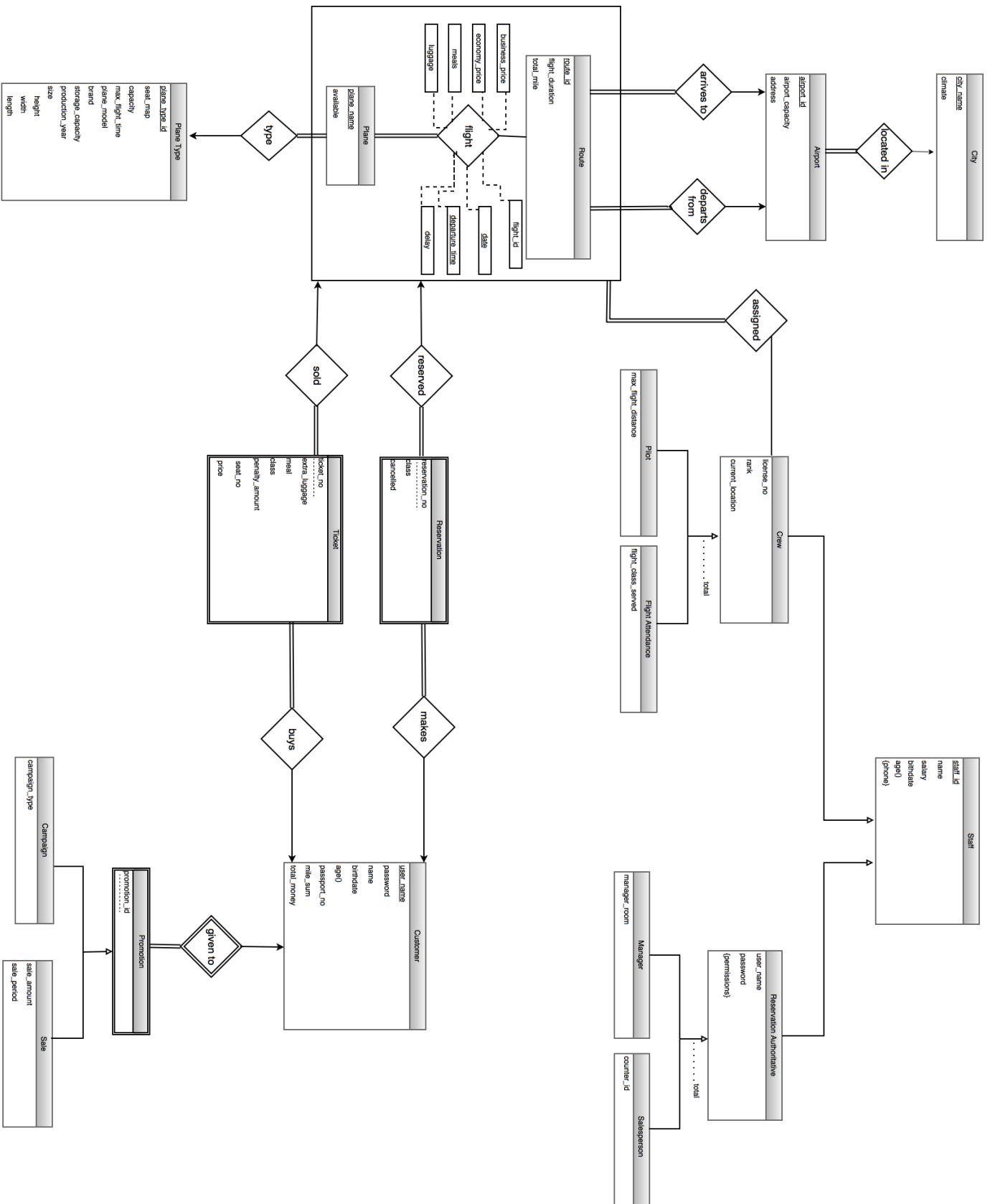
- ✓ Instead of making city an attribute, we represented city as an entity. Besides city name, we keep the climate of the city as well.
- ✓ We removed the 'sold' attribute from the reservation. When a reservation is purchased by the customer a related ticket is created in the system. Instead of a sold 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.

1.2. Revised E/R Diagram



2. RELATION SCHEMAS

2.1. 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) );
```

2.2. 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 );
```

[2.3. 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;
```

[2.4. 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;
```

2.5. 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;
```

2.6. 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;
```

2.7. 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 NOT NULL,  
current_location   varchar(16) NOT NULL,  
FOREIGN KEY (staff_id) REFERENCES staff ) ENGINE = InnoDB;
```

2.8. Pilot

Relational Model:

Pilot(staff_id, max_flight_distance)

Functional Dependencies:

$\text{staff_id} \rightarrow \text{max_flight_distance}$

Candidate Keys:

$\{(\text{staff_id})\}$

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;
```

2.9. Flight Attendance

Relational Model:

Flight_Attendance(staff_id, flight_class_served)

Functional Dependencies:

$\text{staff_id} \rightarrow \text{flight_class_served}$

Candidate Keys:

$\{(\text{staff_id})\}$

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;
```

2.10. 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));

2.11. 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;	

2.12. 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;
```

2.13. 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 );
```

[2.14. Plane](#)**Relational Model:**

Plane(plane_name, available, plane_type_id)

Functional Dependencies:

plane_name \rightarrow 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;
```

2.15. 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;
```

2.16. 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;
```

2.17. Customer

Relational Model:

Customer(user_name, password, name, birthdate, age, passport_no, mile_sum, total_money)

Functional Dependencies:

user_name \rightarrow password, name, birthdate, age, passport_no, mile_sum, total_money

Candidate Keys:

{(user_name)}

Normal Form:

BCNF

Table Definition:

CREATE TABLE customer

```
(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);

```

2.18. 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;
```

2.19. 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;

2.20. 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);
```

2.21. Campaign

Relational Model:

Campaign(user_name, promotion_id, campaign_type)

Functional Dependencies:

user_name, promotion_id \rightarrow 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);
```

2.22. Sale

Relational Model:

Sale(user_name, promotion_id, sale_amount, sale_period)

Functional Dependencies:

user_name, promotion_id \rightarrow 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);
```

3. 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.

4. FUNCTIONAL COMPONENTS

4.1. Use Cases/Scenarios

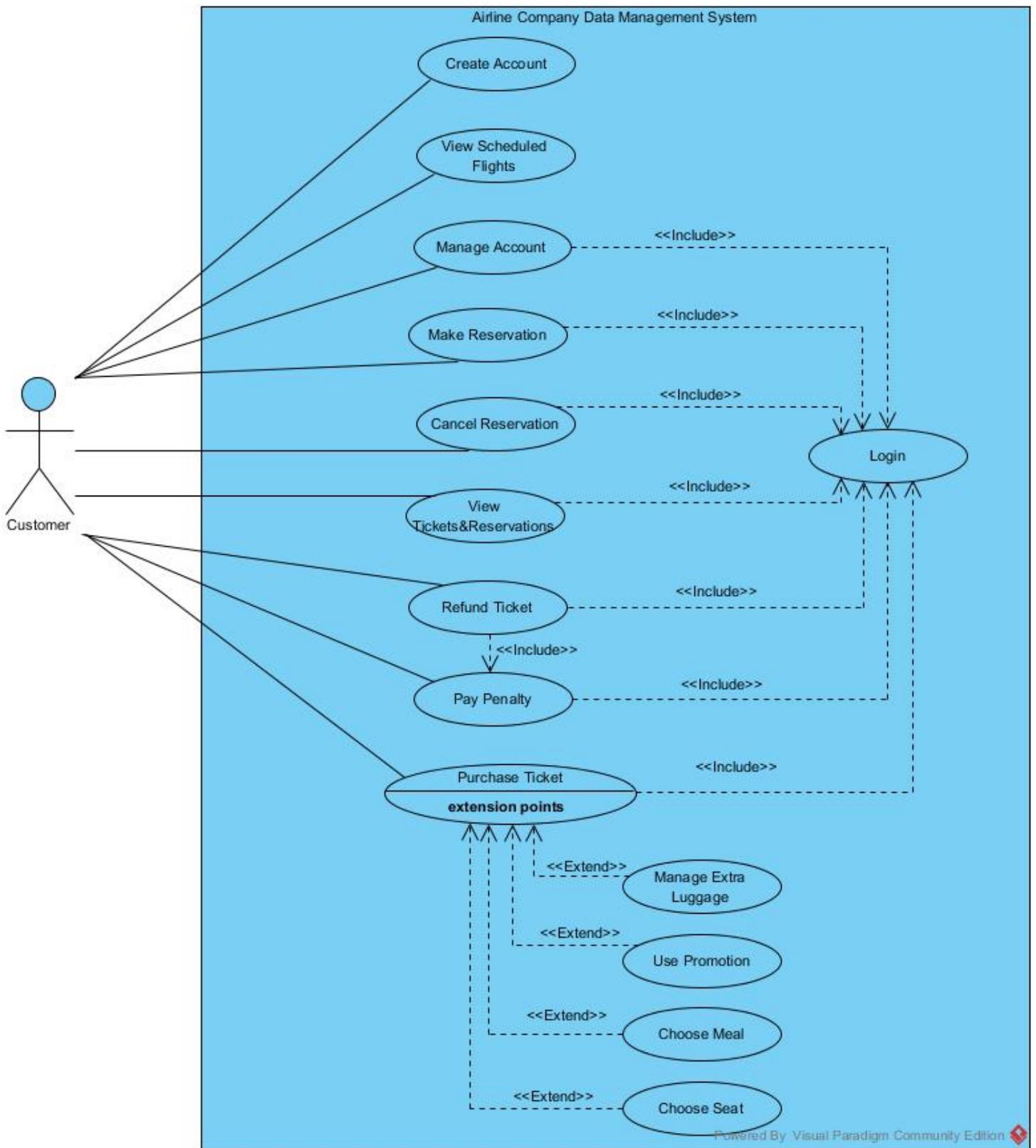
Airline Company Data Management System is responsible from providing reservation services to users along with allowing related managers to control flight, employee, airport, and flight details. The service details of the system vary according to the user. Airline Company Data Management System has 3 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.

4.1.1. Customer Use Cases

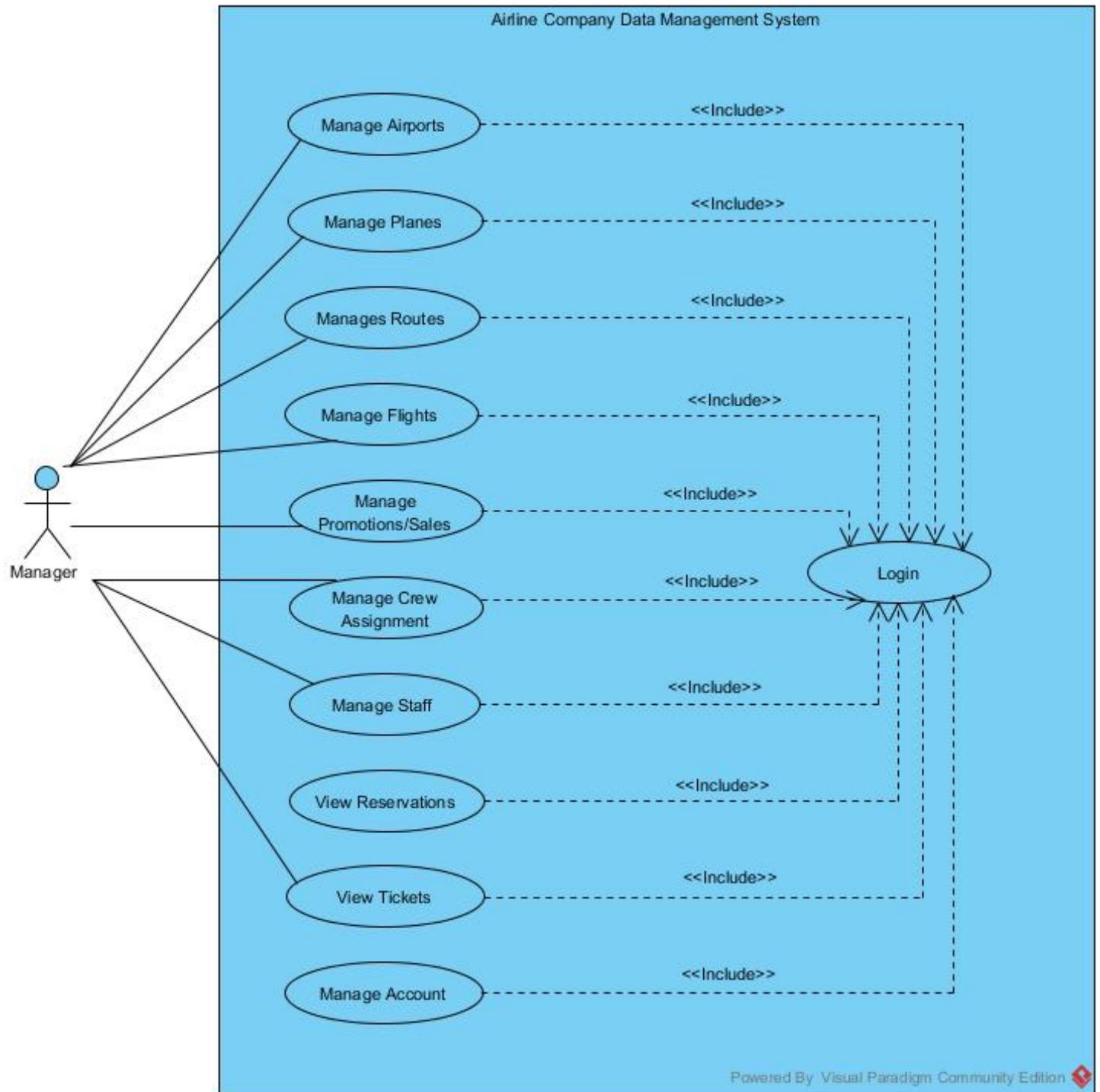


Powered By Visual Paradigm Community Edition

- **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.

4.1.2. Manager Use Cases

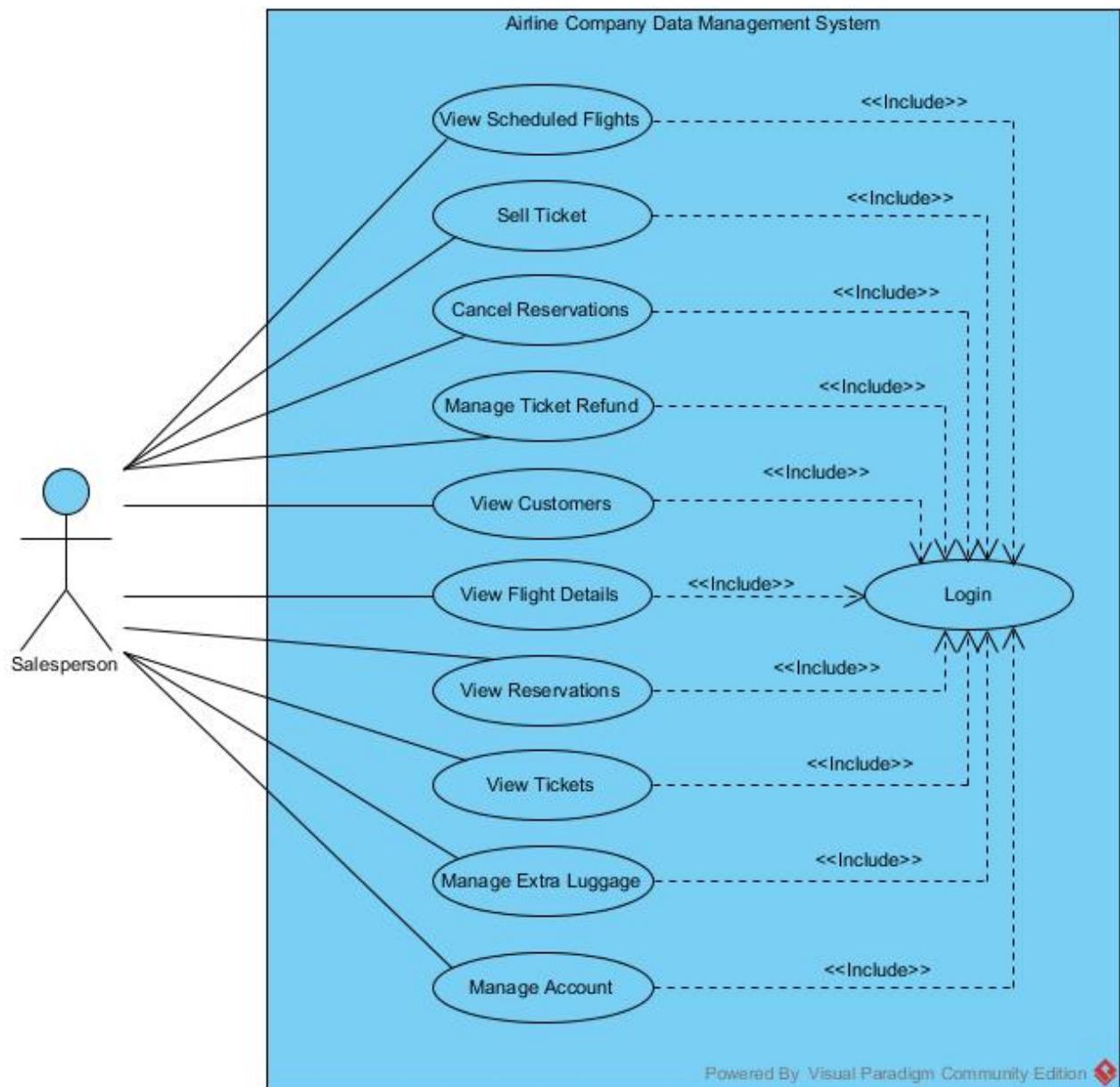


- **Login:** Manager can login to the system with his/her username and password. The manager will be given access to manager functions when the account details are approved. Manager needs to login to the system to be able to perform other operations.

- **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, rise 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.

4.1.3. 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.

4.2. Algorithms

4.2.1. 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.

4.2.2. Promotion Related Algorithms

The system will have promotions to encourage customers to buy 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.

4.2.3. 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 than 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.

4.3. 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.

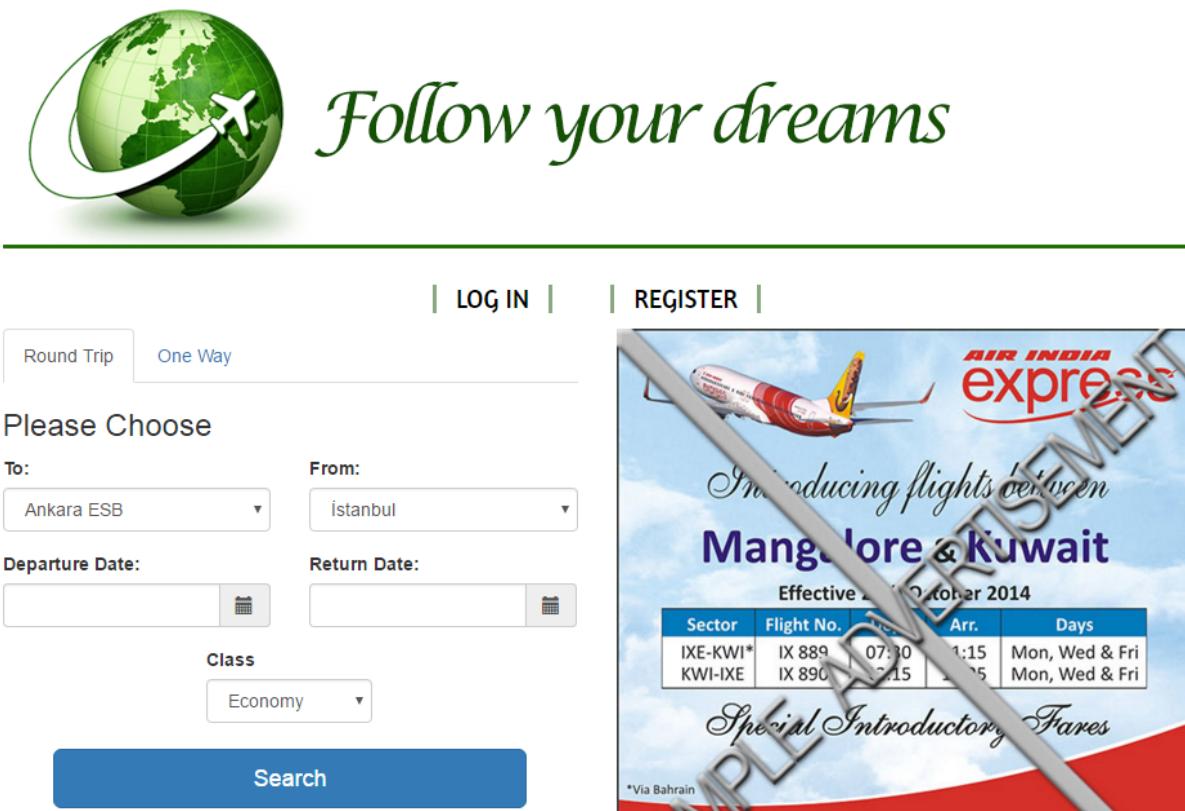
The Interval domain was necessary to keep track of the sale period.

5. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

5.1. Customer User Interface Design

The screens customer can access are described in detail.

5.1.1. 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  
            departureDate = between @departuredate - 2 and @departuredate + 2)  
UNION  
( SELECT flight_id, date, departure_time, price  
   FROM CustomerFlightView C  
  WHERE      departs = @from AND  
            arrives = @to AND  
            departureDate = between @arrivaldate - 2 and @arrivaldate + 2 )
```

5.1.2. Customer Search Flight Screen with One Way Ticket

Please Choose

To: Ankara ESB From: İstanbul

Departure Date:

Class: Economy

Search

Sector	Flight No.	Arr.	Days
IXE-KWI*	IX 889	07:30	1:15
KWI-IXE	IX 890	07:15	05

*Via Bahrain

www.airindiaexpress.in
For reservations and other queries contact us on +965-22438184/85/86
or e-mail kwiairindisales@gmail.com or contact our Authorised Travel Agents.
Please provide your mobile numbers and e-mail id to help Air India Express serve you better.

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:

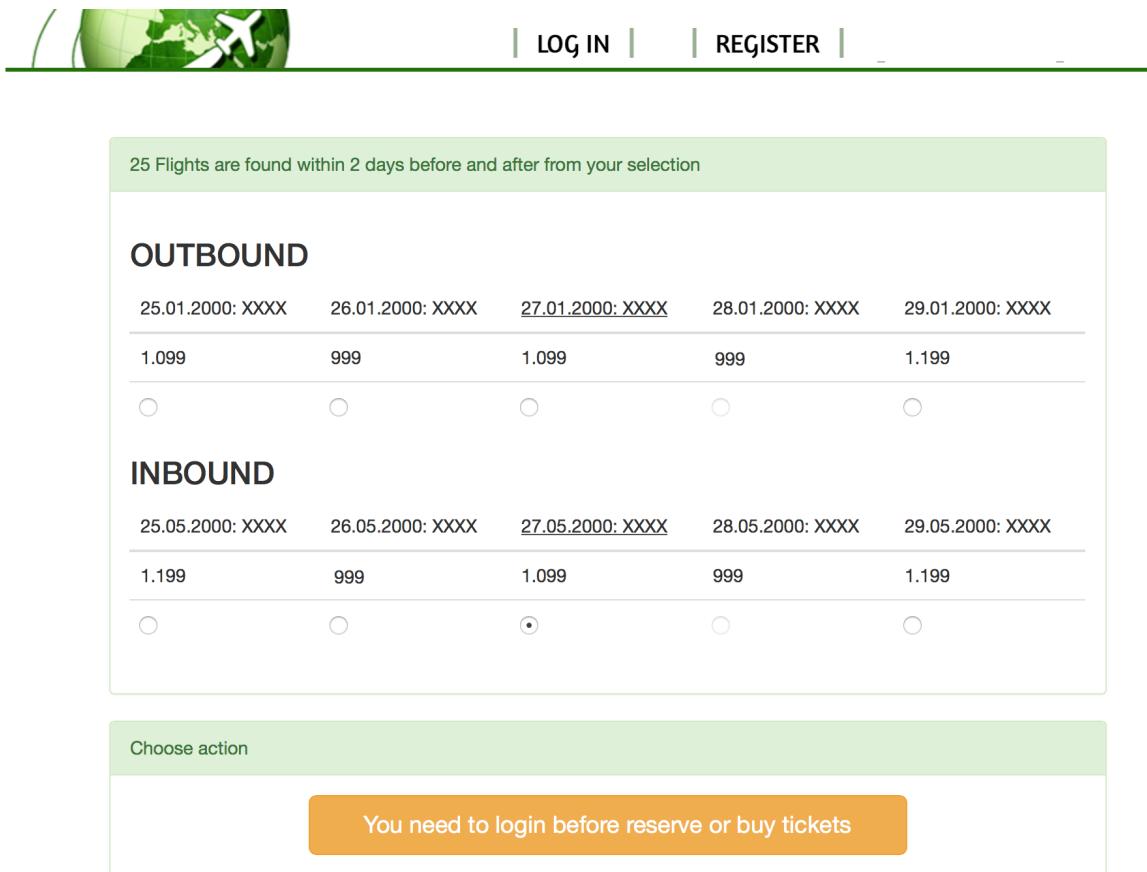
Listing the Airports

```
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 @depaturedate - 2 and @depaturedate + 2
```

5.1.3. Result of Flight Search Screen



The screenshot shows a flight search interface. At the top, there is a logo of a globe with an airplane, followed by 'LOG IN' and 'REGISTER' buttons. A green header bar indicates '25 Flights are found within 2 days before and after from your selection'. Below this, there are two sections: 'OUTBOUND' and 'INBOUND', each listing five flight options with prices and radio buttons for selection. In the 'OUTBOUND' section, the third option (27.01.2000: XXXX) is underlined, and its price (1.099) is bolded. In the 'INBOUND' section, the third option (27.05.2000: XXXX) is underlined, and its price (1.099) is bolded. At the bottom, a green bar says 'Choose action' and an orange button says 'You need to login before reserve or buy tickets'.

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

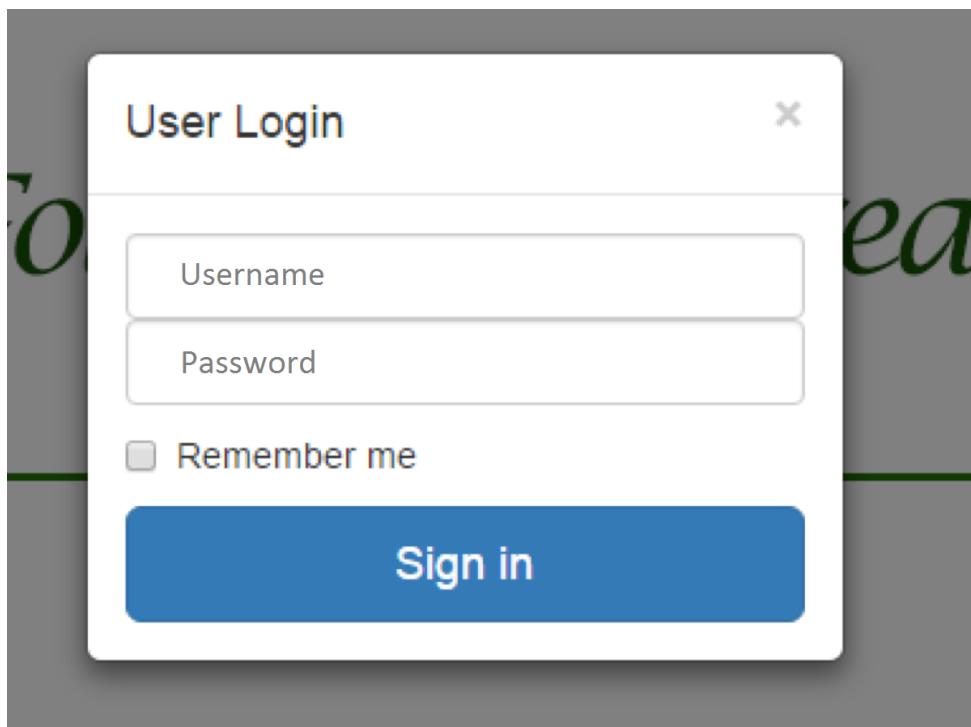
Process: When the user enters flight details in pages 5.1.1. and 5.1.2. and clicks 'Search' button the list of available flights are displayed to him/her. As shown in the above page, the

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.

5.1.4. Login Screen



Inputs: @username, @password

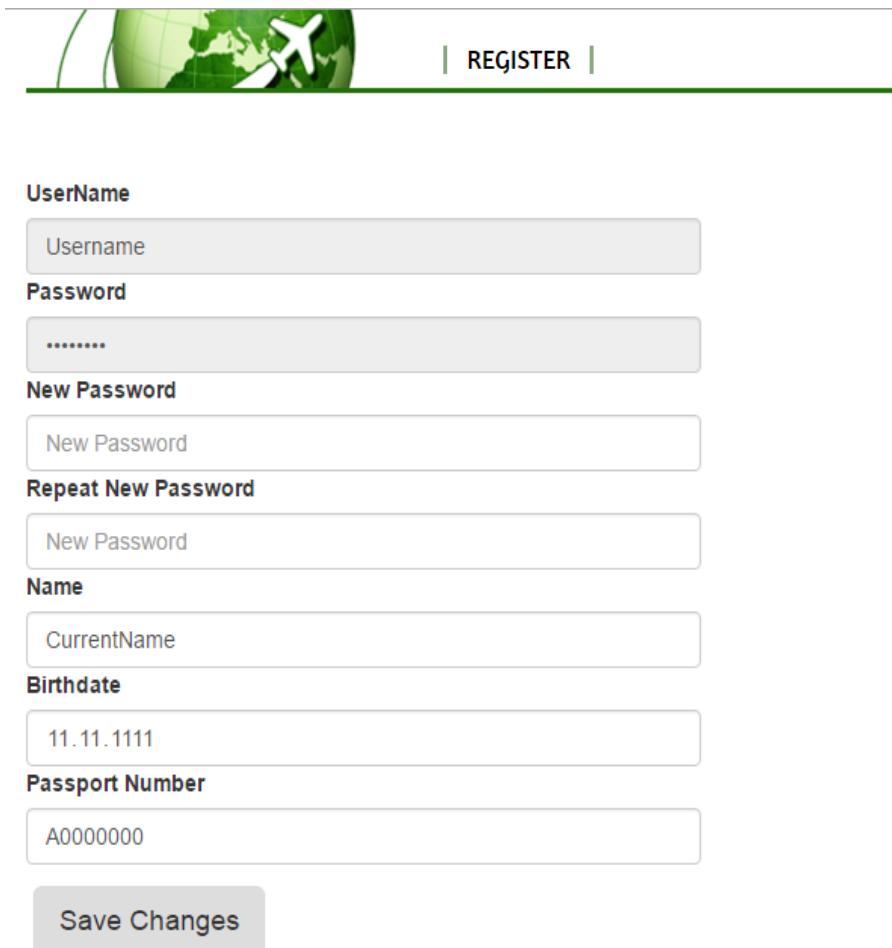
In the homepage, a pop-up window box appears when “Login” is clicked. The box will be asking for access details in order to proceed. Users will be distinguished according to the details they have entered.

SQL Statements:

Login Operation

```
SELECT *  
FROM customer  
WHERE user_name = @username AND  
password = @password
```

5.1.5. Create Account Screen



UserName

Password

New Password

Repeat New Password

Name

Birthdate

Passport Number

Save Changes

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
```

5.1.6. Manage Account Screen

The screenshot shows a web-based application interface for managing a user account. At the top, there is a navigation bar with a globe icon and a flight path graphic on the left, and four menu items on the right: HOME, RESERVATIONS, MY PROFILE (which is highlighted in blue), and LOG OUT. Below the navigation bar is a form for updating user information. The form fields include:

- UserName**: A text input field containing "Username".
- Password**: A text input field containing ".....".
- New Password**: A text input field containing "New Password".
- Repeat New Password**: A text input field containing "New Password".
- Name**: A text input field containing "CurrentName".
- Birthdate**: A text input field containing "11.11.1111".
- Passport Number**: A text input field containing "A0000000".
- Current Mil Sum : 100000**: A text input field containing "100000".

At the bottom of the form are two buttons: "Exit without Changes" and "Save Changes".

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
```

5.1.7. Result of Flight Search Screen for Logged in Customers

The screenshot shows a flight search interface. At the top, there's a navigation bar with a globe icon, followed by links for HOME, RESERVATIONS, MY PROFILE, and LOG OUT. Below the navigation, a green header bar indicates '25 Flights are found within 2 days before and after your selection'. The main content area is divided into two sections: 'OUTBOUND' and 'INBOUND'. Under 'OUTBOUND', there are five flight options listed with their prices: 25.01.2000: XXXX (1.099), 26.01.2000: XXXX (999), 27.01.2000: XXXX (1.099), 28.01.2000: XXXX (----), and 29.01.2000: XXXX (1.199). Each option has a radio button next to it. Under 'INBOUND', there are five flight options listed with their prices: 25.05.2000: XXXX (1.199), 26.05.2000: XXXX (----), 27.05.2000: XXXX (1.099), 28.05.2000: XXXX (999), and 29.05.2000: XXXX (1.199). Each option has a radio button next to it. At the bottom, there's a 'Choose action' section with two buttons: 'Reserve Selection' (orange) and 'Purchase Selection' (green).

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
```

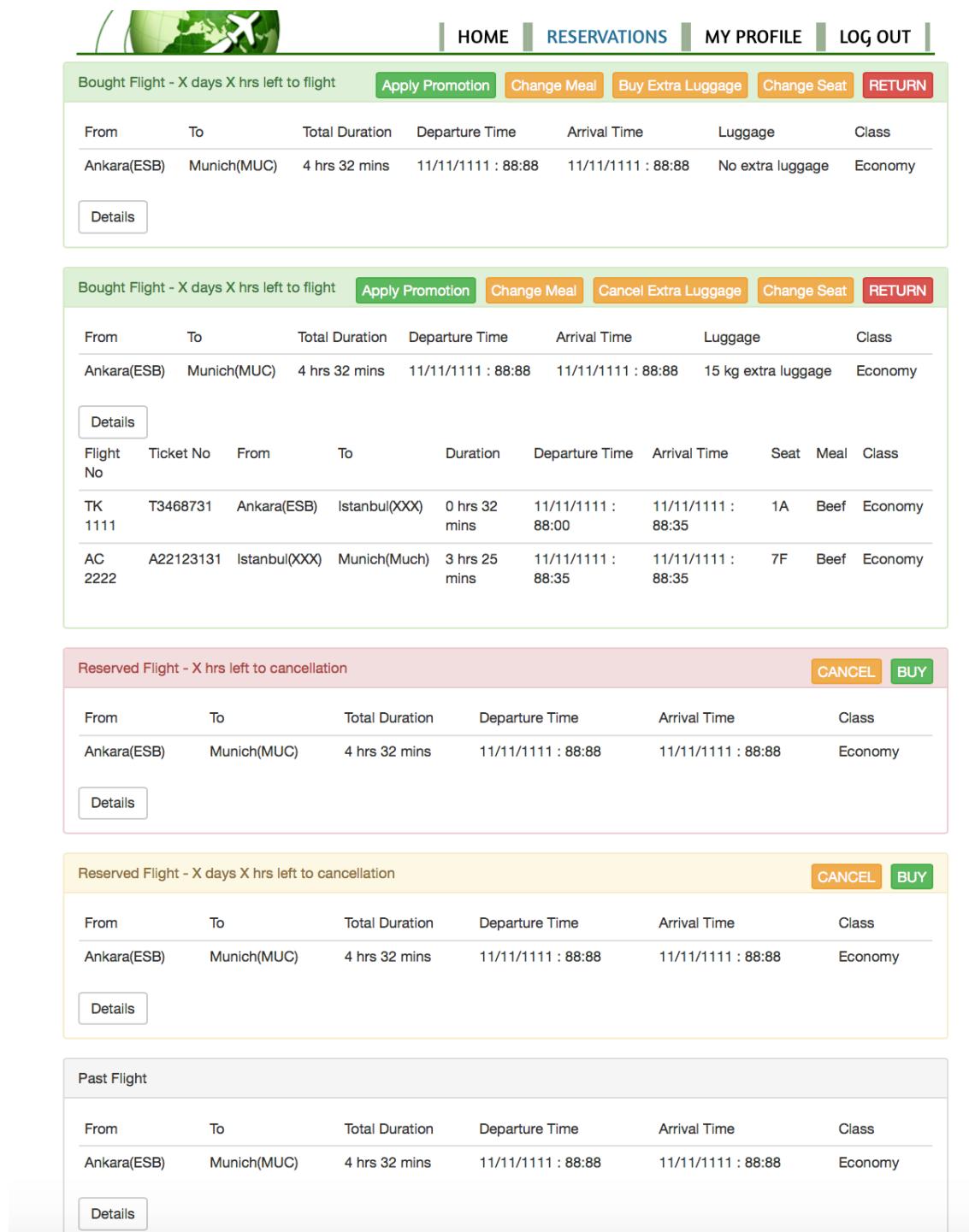
Purchase Ticket

```
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

5.1.8. Customer Reservations & Tickets Screen



The screenshot displays a series of five cards representing different sections of the reservations screen:

- Bought Flight - X days X hrs left to flight:**
 - From: Ankara(ESB) To: Munich(MUC)
 - Total Duration: 4 hrs 32 mins
 - Departure Time: 11/11/1111 : 88:88
 - Arrival Time: 11/11/1111 : 88:88
 - Luggage: No extra luggage
 - Class: Economy
- Reserved Flight - X hrs left to cancellation:**
 - From: Ankara(ESB) To: Munich(MUC)
 - Total Duration: 4 hrs 32 mins
 - Departure Time: 11/11/1111 : 88:88
 - Arrival Time: 11/11/1111 : 88:88
 - Luggage: 15 kg extra luggage
 - Class: Economy
- Past Flight:**
 - From: Ankara(ESB) To: Munich(MUC)
 - Total Duration: 4 hrs 32 mins
 - Departure Time: 11/11/1111 : 88:88
 - Arrival Time: 11/11/1111 : 88:88
 - Luggage: 15 kg extra luggage
 - Class: Economy

Buttons and Links:

- Top navigation: HOME, RESERVATIONS, MY PROFILE, LOG OUT.
- Action buttons: Apply Promotion, Change Meal, Buy Extra Luggage, Change Seat, RETURN.
- Section buttons: Details.
- Reservation buttons: CANCEL, BUY.

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/canceling 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
```

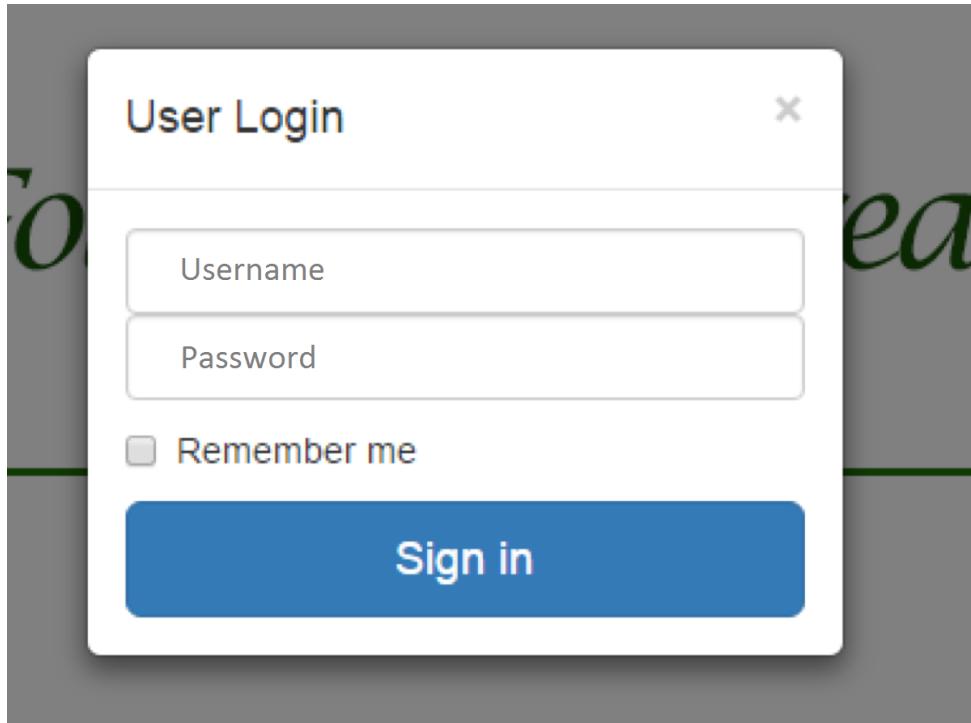
Executes 6.4.1. Mile Sum Trigger After Purchasing Ticket

Executes 6.4.4. Total Money Trigger After Purchasing Ticket

5.2. Manager User Interface Design

The screens manager can access are described in detail.

5.2.1. 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)
```

5.2.2. Manager Manage Account Screen

The screenshot shows a user interface for managing a manager account. At the top, there is a decorative header with a green and blue abstract pattern. To the right of the pattern, the text "Welcome Manager" is displayed in white, and a blue rectangular button labeled "LOG OUT" is positioned next to it. Below the header, the main content area contains several input fields and buttons. The fields are labeled in bold black text: "UserName", "Password", "New Password", "Repeat New Password", "Name", and "Birthdate". Each label is followed by a light gray input field. At the bottom of the form, there are two large, rounded rectangular buttons: "Exit without Changes" on the left and "Save Changes" on the right.

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
```

5.2.3. Manage Airports Screen

Welcome Manager LOG OUT

Airline Corp

- Flights
- Routes
- Crews
- Airports**
- Customers
- Flight Information
- Planes

AIRPORTS

+Add Airport

Code <input type="text" value="withoutspaces,eng chars only"/> City <input type="text" value="eng chars only"/> Airport Capacity <input type="text" value="10"/> Address <input type="text" value="10"/>	Add
---	--

Airport Code	Airport City	Adress	Capacity	Assigned Routes	Assigned Flights		
LAX	Los Angeles	1 World Way,Los Angeles, California 90045	30	16	3	edit	delete
LAX	Los Angeles	1 World Way,Los Angeles, California 90045	30	16	3	edit	delete
LAX	Los Angeles	1 World Way,Los Angeles, California 90045	30	16	3	edit	delete
LAX	Los Angeles	1 World Way,Los Angeles, California 90045	30	16	3	edit	delete
LAX	Los Angeles	1 World Way,Los Angeles, California 90045	30	16	3	edit	delete

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 ands 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.

5.2.4. Manage Airports Screen

Name	Year	Type	Capacity (E,B)	Max Flight Time	Storage Capacity	Status	Send to Repair / Return	Delete
Bayırgülü	2009	Airbus A330	200,30	500	30tons	On Service		
Bayırgülü	2009	Airbus A330	200,30	500	30tons	Out for Repair		
Bayırgülü	2009	Airbus A330	200,30	500	30tons	Out for Repair		
Bayırgülü	2009	Airbus A330	200,30	500	30tons	On Service		
Bayırgülü	2009	Airbus A330	200,30	500	30tons	On Service		

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 ands 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
```

5.2.5. Manage Routes Screen

ROUTES

+Add Route

Route ID	From	To	Duration
R1111	Ankara(ESB)	Munih(MUC)	04:15 <input type="button" value="edit"/> <input type="button" value="x"/>
R1111	Ankara(ESB)	Munih(MUC)	04:15 <input type="button" value="edit"/> <input type="button" value="x"/>
R1111	Ankara(ESB)	Munih(MUC)	04:15 <input type="button" value="edit"/> <input type="button" value="x"/>
R1111	Ankara(ESB)	Munih(MUC)	04:15 <input type="button" value="edit"/> <input type="button" value="x"/>

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 ands 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
```

5.2.6.

5.2.7. Manage Flights Screen

Welcome Manager
[LOG OUT](#)

Airline Corp

- [Flights](#) +Add Flight
- [Routes](#)
- [Crews](#)
- [Airports](#)
- [Customers](#)
- [Flight Information](#)
- [Planes](#)

FLIGHTS

Flight No
ACXXX

Route ID
RXXXXXXX

PlaneName
The Spirit

Meals

 Option 1
 Option 2
 Option 3

Date&Time
2016-01-01

Luggage
10000

Price For Economy
750

Price For Business
1500

Add

FlightNo	Route ID	Plane Name,Type	Meals	Date&Time	Luggage	Prices(E,B)	Status	Crew	Edit	Delete
AC130	R1111 ESB-MUC	The Spirit,A330	B,S,V	01/01/1000 : 0000	15tons	750,1500	On Time			
AC130	R1111 ESB-MUC	The Spirit,A330	B,S,V	01/01/1000 : 0000	15tons	750,1500	On Time			

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 ands 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:

58

Welcome !

Edit Flight

Give Delay

30

Flight No

AC210

Route ID

RXXXXXXX

PlaneName

The Spirit

Meals

Option 1 Option 2 Option 3

Date&Time

2016-01-01

Luggage

10000

Price For Economy

300

Price For Business

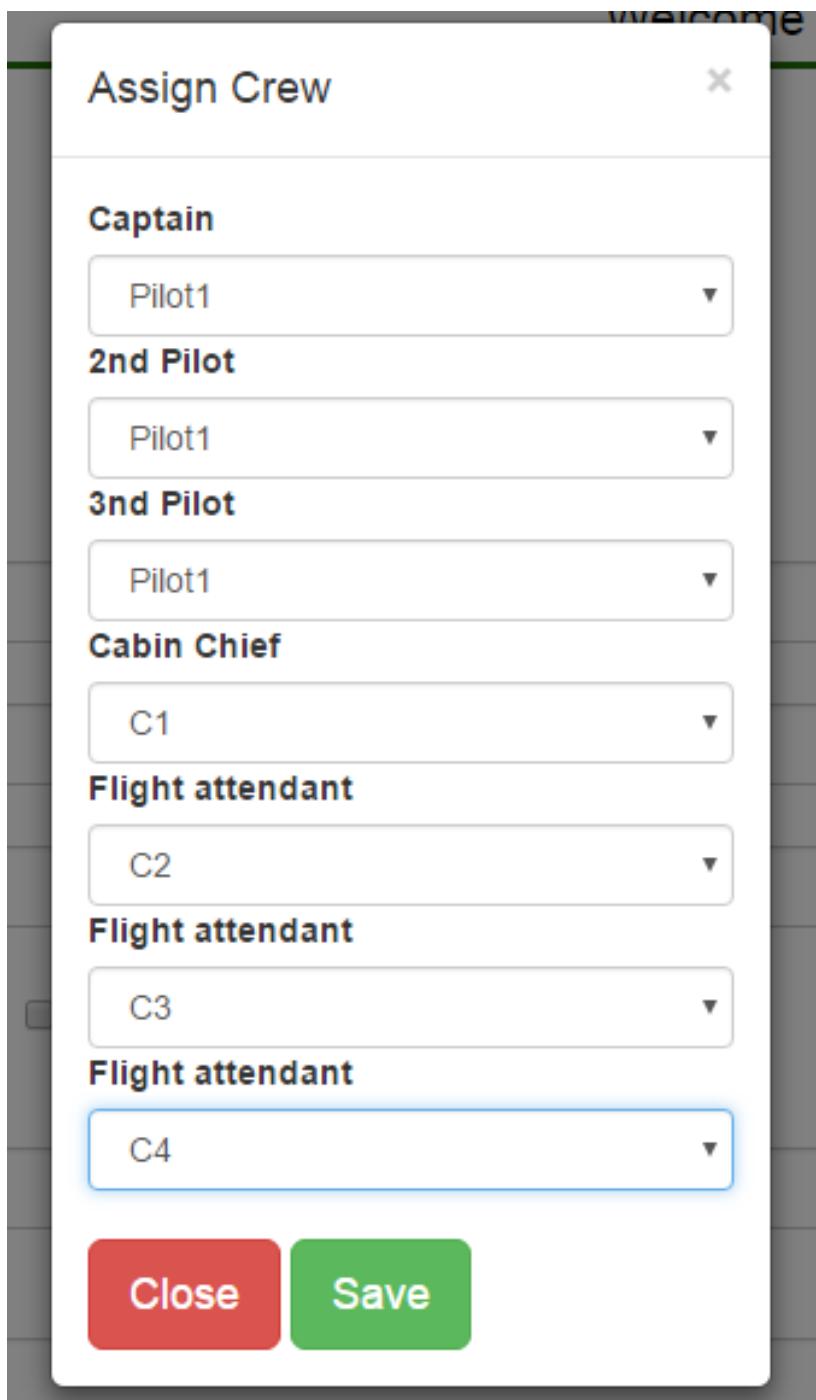
750

Save

The dialog box is titled "Edit Flight". It contains several input fields and checkboxes. The "Give Delay" field has the value "30". The "Flight No" field has the value "AC210". The "Route ID" field has the value "RXXXXXXX". The "PlaneName" field has the value "The Spirit". Under "Meals", there are three checkboxes: "Option 1" (unchecked), "Option 2" (checked), and "Option 3" (unchecked). The "Date&Time" field shows the date "2016-01-01". The "Luggage" field has the value "10000". The "Price For Economy" field has the value "300". The "Price For Business" field has the value "750". At the bottom is a "Save" button.

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.

5.2.8. View Flight Status Screen

Welcome Manager [LOG OUT](#)

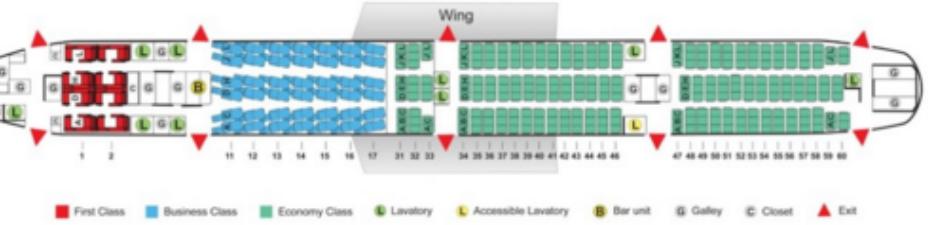
Airline Corp

- [Flights](#)
- [Routes](#)
- [Crews](#)
- [Airports](#)
- [Customers](#)
- [**Flight Information**](#)
- [Planes](#)

Search Flights with time

XX:XX XX/XX/XXXX

Boeing B777-300ER (311 seats)
First Class: Rows 1 - 2: 8 seats Business Class: Rows 11 - 17: 42 seats Economy Class: Rows 31 - 60: 261 seats



■ First Class ■ Business Class ■ Economy Class ● Lavatory ● Accessible Lavatory ● Bar unit ● Galley ● Closet ▲ Exit

Status of Flight: On Time

Reserved / Sold / Total Tickets: 40 -/ 20 -/ 20

Current Reservations for Flight AC2312

More details							
Seat	Ticket NO	Surname	Name	Meal	ExtraLuggage	Reserved/Sold	Delete
1A	4521897	Demir	Oguz	Beef	0	Sold	
1A	4521897	Demir	Oguz	Beef	0	Sold	
1A	4521897	Demir	Oguz	Beef	0	Sold	
1A	4521897	Demir	Oguz	Beef	0	Sold	

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:

Current Sold and Reserved Tickets for Flight AC1111

Reserved Tickets				
Customer Name	Reserved On	Class	Show Customer Info	Delete
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		

Sold Tickets				
Customer Name	Reserved On	Class	Show Customer Info	Delete
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		
oguzdemir	11/11/1111 : 88:88	Economy		

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

5.2.9. Manage Staff Screens

5.2.9.1. Manage Pilot Screen

Welcome Manager [LOG OUT](#)

Airline Corp
 Flights
 Routes
Crews
 Airports
 Customers
 Flight Information
 Planes

Pilots
Flight attendants

+ Add a pilot

StaffId

Name

Salary

Birthdate

Gender

Phone

LicenseNo

Rank

Maximum Flight Distance

Add

StaffID	Name	Salary	BirthDate	Gender	Phone	LicenseNo	Rank	Maximum Flight Distance	Edit	
1	Mehmet Sahin	4000\$	10/01/1994	Male	+9045441345	A1654721	Master	10.000		
1	Mehmet Sahin	4000\$	10/01/1994	Male	+9045441345	A1654721	Master	10.000		

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

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:

StaffID	Name	Salary	Birthdate	Gender	Phone	LicenseNo	Rank	Maximum Flight Distance
1	Mehmet Sahin	4000\$	gg.aa.yyyy	Male/Female	+90XXXXXXXXXX	A1654721	XXX	XX.XXX
1	Mehmet Sahin							
1	Mehmet Sahin							

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

SQL Statements:

View All Pilots

```
SELECT *  
FROM pilot
```

Add a New Pilot

```
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.

5.2.9.2. Manage Flight Attendant Screen

Welcome Manager [LOG OUT](#)

Airline Corp

- [Flights](#)
- [Routes](#)
- [Crews](#)
- [Airports](#)
- [Customers](#)
- [Flight Information](#)
- [Planes](#)

Pilots
Flight attendants

Attendants

[+ Add a attendant](#)

StaffId	<input type="text" value="1111111"/>
Name	<input type="text" value="Sample Attendant"/>
Salary	<input type="text" value="4000\$"/>
Birthdate	<input type="text" value="gg.aa.yyyy"/>
Gender	<input type="text" value="Male/Female"/>
Phone	<input type="text" value="+90XXXXXXXXX"/>
LicenseNo	<input type="text" value="XXXXXXXXXX"/>
Rank	<input type="text" value="XXX"/>
Serving Class	<input type="text" value="Economy/Business/First"/>

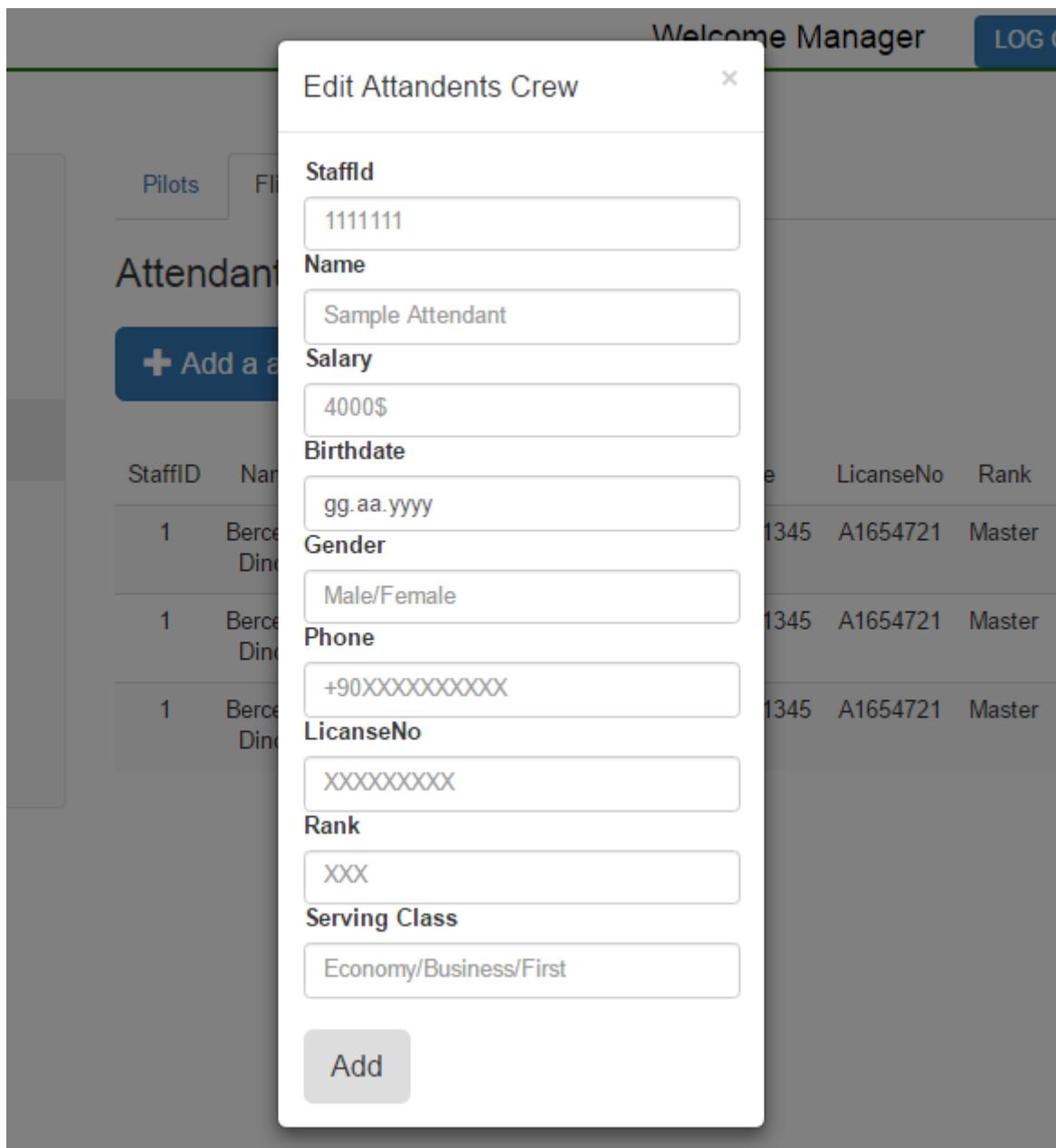
Add

StaffID	Name	Salary	BirthDate	Gender	Phone	LicenseNo	Rank	Economy	Edit
1	Berreste Dincer	3000\$	10/01/1994	Female	+9045441345	A1654721	Master	Economy	
1	Berreste Dincer	3000\$	10/01/1994	Female	+9045441345	A1654721	Master	Economy	

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 “Save”, 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.

5.2.9.3. 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.

5.2.10. View Customers Screen

Welcome Manager LOG OUT

Airline Corp

Flights

Routes

Crews

Airports

Customers

Flight Information

Planes

CUSTOMERS

>Show Customers with + 10.000 miles

Username	Name	Reservations	
ogzdemr	Oguz		

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:

Info of User xxx with name xxx

Bought Flight - X days X hrs left to flight						
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	No extra luggage	Economy
Details						

Bought Flight - X days X hrs left to flight						
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	15 kg extra luggage	Economy
Details						

Reserved Flight - X hrs left to cancellation						
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88		Economy
Details						

Reserved Flight - X days X hrs left to cancellation						
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88		Economy
Details						

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

SQL Statements:

View All Customers

```
SELECT *
FROM ManagerCustomerView
```

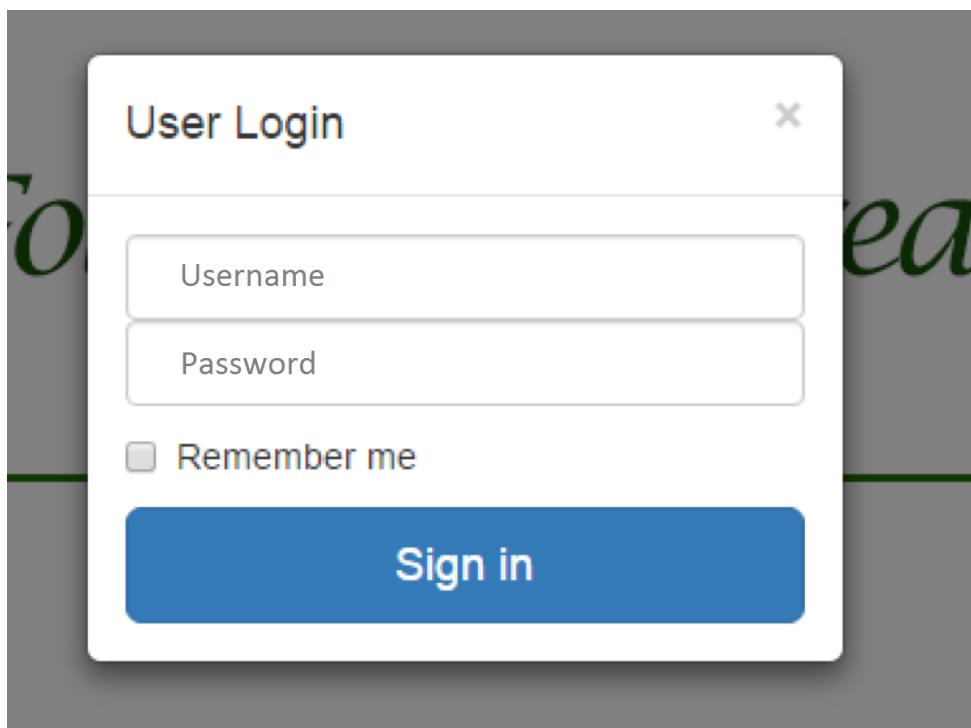
View Tickets and Reservations Of Customer

```
(SELECT date, time, departs, arrives, flight_duration, class
FROM CustomerReservationView) UNION
(SELECT date, time, departs, arrives, flight_duration, extra_luggage, class
FROM CustomerTicketView)
```

5.3. Salesperson User Interface Design

The screens salesperson can access are described in detail.

5.3.1. 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:

```
SELECT *
FROM reservation_authority
WHERE user_name = @username AND
      password = @password AND
      user_name IN (SELECT user_name FROM reservation_authority NATURAL JOIN
salesperson)
```

5.3.2. 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.

5.3.3. 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.

5.3.4. Salesperson Customer View

Username	Name	Reservations
ogzdemr	Oguz	

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
```

5.3.5. Salesperson Customer Reservations & Tickets View

[LOG OUT](#)

Info of User xxx with name xxx

Bought Flight - X days X hrs left to flight							Apply Promotion	Change Meal	Buy Extra Luggage	Change Seat	RETURN	
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class						
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	No extra luggage	Economy						
Details												

Bought Flight - X days X hrs left to flight							Apply Promotion	Change Meal	Cancel Extra Luggage	Change Seat	RETURN																																											
From	To	Total Duration	Departure Time	Arrival Time	Luggage	Class																																																
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	15 kg extra luggage	Economy																																																
Details																																																						
<table border="1"> <thead> <tr> <th>Flight No</th> <th>Ticket No</th> <th>From</th> <th>To</th> <th>Duration</th> <th>Departure Time</th> <th>Arrival Time</th> <th>Seat</th> <th>Meal</th> <th>Class</th> <th colspan="4"></th> </tr> </thead> <tbody> <tr> <td>TK 1111</td> <td>T3468731</td> <td>Ankara(ESB)</td> <td>Istanbul(XXX)</td> <td>0 hrs 32 mins</td> <td>11/11/1111 : 88:00</td> <td>11/11/1111 : 88:35</td> <td>1A</td> <td>Beef</td> <td>Economy</td> <td colspan="4"></td></tr> <tr> <td>AC 2222</td> <td>A22123131</td> <td>Istanbul(XXX)</td> <td>Munich(Much)</td> <td>3 hrs 25 mins</td> <td>11/11/1111 : 88:35</td> <td>11/11/1111 : 88:35</td> <td>7F</td> <td>Beef</td> <td>Economy</td> <td colspan="4"></td></tr> </tbody> </table>													Flight No	Ticket No	From	To	Duration	Departure Time	Arrival Time	Seat	Meal	Class					TK 1111	T3468731	Ankara(ESB)	Istanbul(XXX)	0 hrs 32 mins	11/11/1111 : 88:00	11/11/1111 : 88:35	1A	Beef	Economy					AC 2222	A22123131	Istanbul(XXX)	Munich(Much)	3 hrs 25 mins	11/11/1111 : 88:35	11/11/1111 : 88:35	7F	Beef	Economy				
Flight No	Ticket No	From	To	Duration	Departure Time	Arrival Time	Seat	Meal	Class																																													
TK 1111	T3468731	Ankara(ESB)	Istanbul(XXX)	0 hrs 32 mins	11/11/1111 : 88:00	11/11/1111 : 88:35	1A	Beef	Economy																																													
AC 2222	A22123131	Istanbul(XXX)	Munich(Much)	3 hrs 25 mins	11/11/1111 : 88:35	11/11/1111 : 88:35	7F	Beef	Economy																																													

Reserved Flight - X hrs left to cancellation							CANCEL	BUY
From	To	Total Duration	Departure Time	Arrival Time	Class			
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	Economy			
Details								

Reserved Flight - X days X hrs left to cancellation							CANCEL	BUY
From	To	Total Duration	Departure Time	Arrival Time	Class			
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	Economy			
Details								

Past Flight						
From	To	Total Duration	Departure Time	Arrival Time	Class	
Ankara(ESB)	Munich(MUC)	4 hrs 32 mins	11/11/1111 : 88:88	11/11/1111 : 88:88	Economy	

Inputs: @curtomernome (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

6. ADVANCED DATABASE COMPONENTS

6.1. Views

6.1.1. 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 )
```

6.1.2. 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)
```

6.1.3. 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

```

6.1.4. 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)

```

6.1.5. 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 )

```

6.2. 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.

6.2.1. 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.

6.2.2. 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.

6.2.3. 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.

6.2.4. 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.

6.2.5. 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.

6.2.6. 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.

6.2.7. 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.

6.2.8. 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.

6.3. Reports

6.3.1. 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

```

6.3.2. 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

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

```

6.3.3. 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, salespersons

```

6.3.4. 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

```

6.3.5. 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
```

6.4. Triggers

6.4.1. 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.

6.4.2. 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.

6.4.3. Cancel Reservation Trigger After Refunding Ticket

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

6.4.4. 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.

6.4.5. 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.

6.4.6. 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.

6.4.7. 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.

6.4.8. 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.

6.4.9. 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.

6.4.10. 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.

6.5. Constraints

6.5.1. Minimum Salary Constraint for Staff

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

6.5.2. 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.

6.5.3. Simultaneous Flights Constraint for Customer Flights

Ticket cannot have a pair of exact same date and departure_time values for the same customer.

6.5.4. Extra Luggage Constraint for Ticket

Ticket cannot have a more than two extra_luggage.

6.5.5. 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.

6.5.6. 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.

6.5.7. 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)

6.5.8. 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.

6.5.9. Pilot Number Constraint for Pilot Assignment

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

6.5.10. 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.

6.5.11. Total Money Constraint for Customer

The total_money attribute of customer cannot be below 0.

7. 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.