

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 4](#_Toc447019951)

[1.1. Changes Made in the E/R Diagram 4](#_Toc447019952)

[1.2. Revised E/R Diagram 6](#_Toc447019953)

[2. RELATION SCHEMAS 7](#_Toc447019954)

[2.1. Staff 7](#_Toc447019955)

[2.2. Reservation Authoritative 7](#_Toc447019956)

[2.3. Permissions 8](#_Toc447019957)

[2.4. Manager 8](#_Toc447019958)

[2.5. Salesperson 9](#_Toc447019959)

[2.6. Ticketing/Gate Agent 9](#_Toc447019960)

[2.7. Crew 10](#_Toc447019961)

[2.8. Pilot 11](#_Toc447019962)

[2.9. Flight Attendance 11](#_Toc447019963)

[2.10. City 12](#_Toc447019964)

[2.11. Airport 12](#_Toc447019965)

[2.12. Route 13](#_Toc447019966)

[2.13. Plane Type 14](#_Toc447019967)

[2.14. Plane 14](#_Toc447019968)

[2.15. Flight 15](#_Toc447019969)

[2.16. Flight Crew 16](#_Toc447019970)

[2.17. Customer 16](#_Toc447019971)

[2.18. Memberships 17](#_Toc447019972)

[2.19. Reservation 18](#_Toc447019973)

[2.20. Ticket 18](#_Toc447019974)

[2.21. Promotion 19](#_Toc447019975)

[2.22. Campaign 20](#_Toc447019976)

[2.23. Sale 20](#_Toc447019977)

[3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES 22](#_Toc447019978)

[4. FUNCTIONAL COMPONENTS 23](#_Toc447019979)

[4.1. Use Cases/Scenarios 23](#_Toc447019980)

[4.1.1. Customer Use Cases 23](#_Toc447019981)

[4.1.2. Manager Use Cases 26](#_Toc447019982)

[4.1.3. Salesperson Use Cases 29](#_Toc447019983)

[4.1.4. Ticketing/Gate Agent Use Cases 32](#_Toc447019984)

[4.2. Algorithms 33](#_Toc447019985)

[4.2.1. Ticket Related Algorithms 33](#_Toc447019986)

[4.2.2. Promotion Related Algorithms 34](#_Toc447019987)

[4.2.3. Reservation Related Algorithms 34](#_Toc447019988)

[4.3. Data Structures 35](#_Toc447019989)

[5. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS 36](#_Toc447019990)

[6. ADVANCED DATABASE COMPONENTS 37](#_Toc447019991)

[6.1. Views 37](#_Toc447019992)

[6.1.1. Customer Flight View 37](#_Toc447019993)

[6.1.2. Customer Reservation View 37](#_Toc447019994)

[6.1.3. Customer Flight Details View 37](#_Toc447019995)

[6.1.4. Customer Ticket View 37](#_Toc447019996)

[6.1.5. Manager Customer View 38](#_Toc447019997)

[6.1.6. Salesperson/Agent Flight Details View 38](#_Toc447019998)

[6.2. Stored Procedures 38](#_Toc447019999)

[6.2.1. Make Reservation Stored Procedure 38](#_Toc447020000)

[6.2.2. Cancel Reservation Stored Procedure 39](#_Toc447020001)

[6.2.3. Purchase Ticket Stored Procedure 39](#_Toc447020002)

[6.2.4. Refund Ticket Stored Procedure 39](#_Toc447020003)

[6.2.5. Delete Airport Procedure 40](#_Toc447020004)

[6.2.6. Delete Route Procedure 40](#_Toc447020005)

[6.2.7. Cancel Flight Procedure 40](#_Toc447020006)

[6.2.8. Send Plane to Repair Stored Procedure 41](#_Toc447020007)

[6.3. Reports 41](#_Toc447020008)

[6.3.1. Total Number of Customers Registered to the System, Total Number of Customers with Reservation, Total Number of Customers with Tickets 41](#_Toc447020009)

[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 42](#_Toc447020010)

[6.3.3. Total Number of Employees in Each Role and the Average Salary of Each Role 42](#_Toc447020011)

[6.3.4. Total Number of Flights Associated with Each Airport, the Cities of the Airports, and the Number of Customers Using Each Airport 43](#_Toc447020012)

[6.4. Triggers 43](#_Toc447020013)

[6.5. Constraints 44](#_Toc447020014)

# Revised E/R Model

## Changes Made in the E/R Diagram

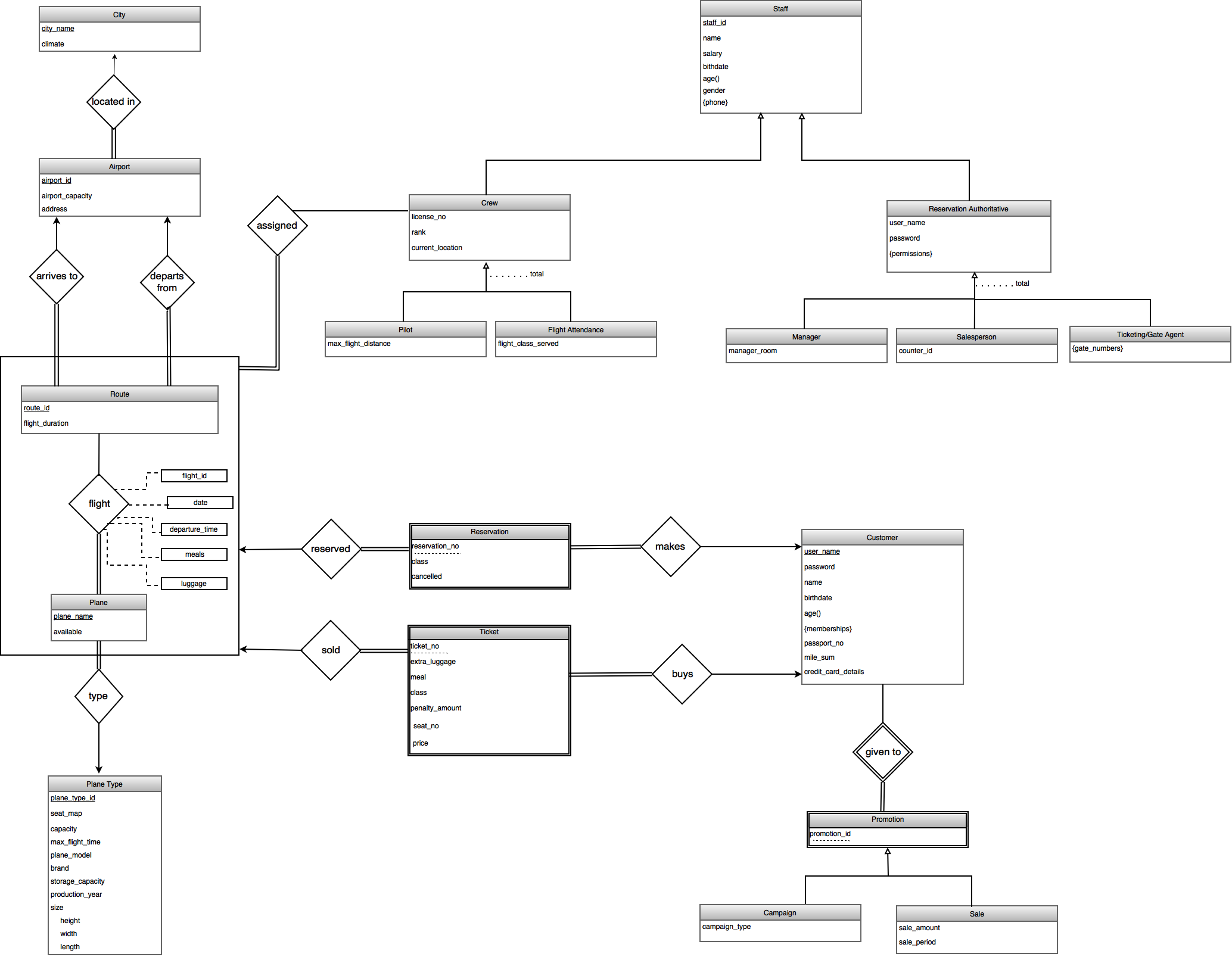
After we received feedback from pour assistant, we made the following changes in our E/R model in order to provide a better database structure for our project:

* Instead of making city an attribute, we represented city as an entity. Besides city name, we keep the climate of the city as well in order to allow customers to view the climate of the cities they are travelling to.
* 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\_numbers to Ticketing/Gate Agent in order to track which gate the employee is currently assigned to. We used a multivalued attribute since a ticketing/gate agent can be responsible from controlling multiple gates.

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:

* We added price attribute to ticket to track the price that is paid for each ticket.
* We added credit\_card\_details attribute to the customer in order to perform purchasing operations easily.
* We added meals and luggage attributes to flight relation in order to keep track of the total luggage registered to the flight and the total number of meals bought.
* 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.

## Revised E/R Diagram



# RELATION SCHEMAS

## Staff

**Relational Model:**

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

**Functional Dependencies:**

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

**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),

phone varchar(15) NOT NULL );

## Reservation Authoritative

**Relational Model:**

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

**Functional Dependencies:**

staff\_id -> user\_name, password

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

create table reservation\_authoritative

(staff\_id int PRIMARY KEY,

user\_name varchar(32) NOT NULL,

password varchar(32) NOT NULL,

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

## Permissions

**Relational Model:**

Permissions (staff\_id, permission\_name)

**Functional Dependencies:**

staff\_id -> permission\_name

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE permissions

(staff\_id int PRIMARY KEY,

permission\_name varchar(32),

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

## Manager

**Relational Model:**

Manager(staff\_id, manager\_room)

**Functional Dependencies:**

staff\_id -> manager\_room

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE manager

(staff\_id int PRIMARY KEY,

manager\_room varchar(4)

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

## Salesperson

**Relational Model:**

Salesperson(staff\_id, counter\_id)

**Functional Dependencies:**

staff\_id -> counter\_id

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE salesperson

(staff\_id int PRIMARY KEY,

counter\_id int

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

## Ticketing/Gate Agent

**Relational Model:**

Ticketing\_Gate\_Agent(staff\_id, gate\_number)

**Functional Dependencies:**

staff\_id -> gate\_number

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE ticketing\_gate\_agent

(staff\_id int PRIMARY KEY,

gates varchar(512)

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

## Crew

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE crew

(staff\_id int PRIMARY KEY,

license\_no char(10) NOT NULL,

rank int NOL NULL,

current\_location varchar(16) NOT NULL,

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

d

## Pilot

**Relational Model:**

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

**Functional Dependencies:**

staff\_id -> max\_flight\_distance

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE pilot

(staff\_id int PRIMARY KEY,

max\_flight\_distance int NOT NULL

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

## Flight Attendance

**Relational Model:**

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

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE flight\_attendance

(staff\_id int PRIMARY KEY,

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

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

## City

**Relational Model:**

City(city\_name, climate)

**Functional Dependencies:**

city\_name -> climate

**Candidate Keys:**

{(city\_name)}

**Normal Form:**

BCNF

**Table Definition:**

**CREATE TABLE** city

(city\_name varchar(20) PRIMARY KEY,

climate char(1));

d

## Airport

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(airport\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE airport

(airport\_id int PRIMARY KEY,

airport\_capacity int NOT NULL,

address varchar(100),

city\_name varchar(20),

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

## Route

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(route\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE route

(route\_id int PRIMARY KEY,

flight\_duration time NOT NULL,

departs int NOT NULL,

arrives int NOT NULL,

foreign key(departs) REFERENCES airport,

foreign key(arrives) REFERENCES airport) ENGINE InnoDB;

## Plane Type

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(plane\_type\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE plane\_type

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

seatmap varchar(200) NOT NULL,

capacity int NOT NULL,

max\_flight\_time time NOT NULL,

plane\_model varchar(4),

brand varchar(10),

storage\_capacity int NOT NULL,

production\_year int,

width int,

height int,

length int );

## Plane

**Relational Model:**

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

**Functional Dependencies:**

plane\_name -> plane\_type\_id

**Candidate Keys:**

{(plane\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE plane

(plane\_name varchar(20) PRIMARY KEY

foreign key(plane\_type\_id) REFERENCES plane\_type) ENGINE InnoDB;

## Flight

**Relational Model:**

Flight(plane\_name, route\_id, flight\_id, date, departure\_time, meals, luggage)

**Functional Dependencies:**

plane\_name, route\_id -> flight\_id, date, departure\_time, meals, luggage

**Candidate Keys:**

{(plane\_name, route\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**CREATE TABLE** flight

(plane\_name varchar(20) PRIMARY KEY,

route\_id int PRIMARY KEY,

flight\_id int NOT NULL,

date date NOT NULL,

departure\_time time,

meals int,

luggage int,

foreign key (route\_id) REFERENCES route,

foreign key (plane\_name) REFRENCES plane) ENGINE = InnoDB;

## Flight Crew

**Relational Model:**

Flight\_Crew(staff\_id, plane\_name, route\_id)

**Functional Dependencies:**

staff\_id -> plane\_name, route\_id

**Candidate Keys:**

{(plane\_name, route\_id)}

**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,

foreign key (route\_id) REFERENCES route,

foreign key (staff\_name) REFERENCES staff,

foreign key (plane\_name) REFRENCES plane) ENGINE = InnoDB;

## Customer

**Relational Model:**

Customer(user\_name, password, name, birthdate, age, passport\_no, mile\_sum, crecit\_card\_details)

**Functional Dependencies:**

user\_name -> password, name, birthdate, age, passport\_no, mile\_sum, credit\_card\_details

**Candidate Keys:** {(user\_name)}

Table Definition:

**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

credit\_card\_details varchar(32) );

## Memberships

**Relational Model:**

Memberships (user\_name, membership\_name)

**Functional Dependencies:**

user\_name -> membership\_name

**Normal Form:**

BCNF

**Candidate Keys:**

{(user\_name)}

**Table Definition:**

CREATE TABLE memberships

(user\_name varchar(32) PRIMARY KEY,

membership\_name varchar(32),

FOREIGN KEY (user\_name) REFERENCES customer) ENGINE = InnoDB;

## Reservation

**Reservation**

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(user\_name, plane\_name, route\_id, 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,

class char(1) NOT NULL,

cancelled char(1) NOT NULL,

foreign key (user\_name) REFERENCES customer,

foreign key (route\_id) REFERENCES route,

foreign key (plane\_name) REFRENCES plane) ENGINE = InnoDB;

## Ticket

**Relational Model:**

Ticket( user\_name, plane\_name, route\_id, ticket\_no, price, class, meal, extra\_luggage, seat\_no)

**Functional Dependencies:**

user\_name, plane\_name, route\_id, ticket\_no -> price, class, meal, extra\_luggage, seat\_no

**Candidate Keys:**

{(user\_name, plane\_name, route\_id, 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,

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 (route\_id) REFERENCES route,

foreign key (plane\_name) REFRENCES plane) ENGINE = InnoDB;

## Promotion

**Relational Model:**

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

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

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

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

foreign key user\_name REFERENCES customer);

## Campaign

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**create table** campaign

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

campaign\_type varchar(10) NOT NULL,

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

## Sale

**Relational Model:**

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

**Functional Dependencies:**

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

**Candidate Keys:**

{(user\_name, promotion\_id)}

**Normal Form:**

BCNF

**Table Definition:**

**CREATE TABLE** sale

(user\_name varchar(32) PRIMARY KEY,

promotion\_id int PRIMARY KEY,

sale\_amount int,

sale\_period interval,

foreign key (user\_name, promotion\_id) REFERENCES given\_promotion);

# FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

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

# FUNCTIONAL COMPONENTS

## Use Cases/Scenarios

Airline Company Data Management System is responsible from providing reservation services to users along with allowing related managers to control flight, employee, airport, and flight details. The service details of the system vary according to the user. Airline Company Data Management System has 4 users: Customer, Manager, Salesperson, and Ticketing/Gate Agent. Even though some services are common to all users, each user is allowed to access different functionalities of the system.

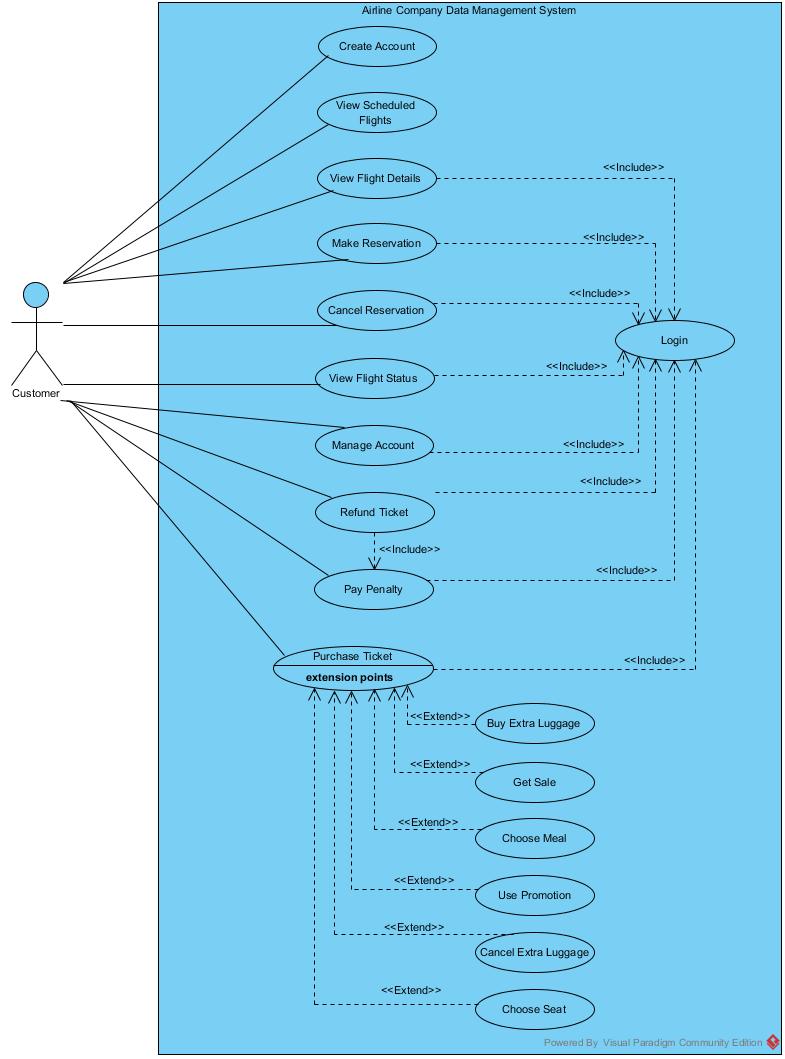
Customer is able to perform actions related to reservations such as viewing flight details and making payment.

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.

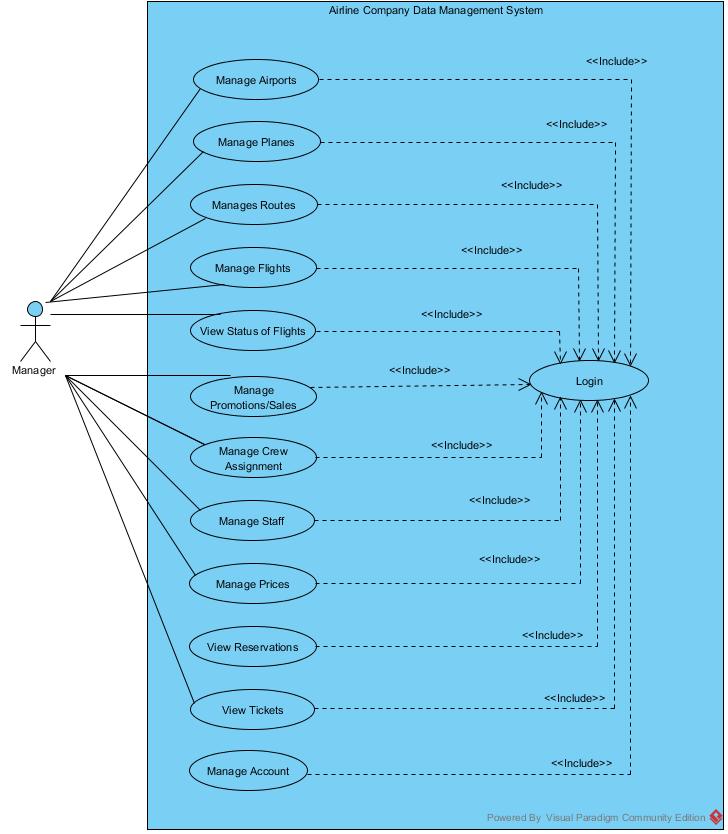
Ticketing/Gate Agent is responsible from finalizing flight details for customers.

### Customer Use Cases



* **Create Account:** The customers can create an account in the system by specifying their unique username, password, name, birthdate, credit card details, and membership details. The user can use this account information to login to the system for reservations.
* **Login:** The customers can login to the system using his/her username and password in order to be able make reservation/purchasing operations.
* **View Scheduled Flights:** The customers can view all flights that are registered to system. The user can specify date, arrival city and airport, and departure city and airport in order to view the flights he/she is interested in.
* **View Flight Details:** The customers can view flight details such as classes available, meals available, seat plan, and price. The customer is required to login to the system in order to be able to view flight details.
* **Make Reservation:** The customers can make reservations to the available flights by indicating the number and status of the passengers (child, student, etc.). The customers can also choose either one-way or return tickets. The customers need to login to the system to be able to make reservations.
* **Cancel Reservation:** The customers can cancel their reservations if the system allows them to. The customers need to login to the system to be able to cancel their reservations.
* **View Flight Status:** The customers can view the status of the flights they made reservation or purchased ticket for. The status information includes meals bought, class choice, seat choice, and delay amount of the flight. The customers need to login to the system to be able to view the status of their flights.
* **Manage Account:** The customers can update their account information. The username and password details, credit card details, and, membership information can be changed or deleted by the customer when it is necessary. The customers need to login to the system to be able to update their account details.
* **Refund Ticket:** The customers can refund their tickets if the system allows. They can cancel their ticket and their money will be refunded. The Refund Ticket use case includes paying the penalty. The customers need to login to the system to be able to refund their tickets.
* **Pay Penalty:** The customers can pay the penalty amount from their account when they refund a ticket. The customers need to login to the system to be able to pay the penalty.
* **Purchase Ticket:** The customers can purchase tickets from the flights they have reservation from. The date, airports, and number of passengers need to be indicated before purchase. The customers need to login to the system to be able to purchase ticket.
* **Buy Extra Luggage:** The customers can buy extra luggage for the flights they bought a ticket from. They can indicate the extra luggage amount and make the necessary payment.
* **Cancel Extra Luggage:** The customers can cancel the extra luggage they bought and their money will be refunded.
* **Choose Seat:** The customers can choose their seats for the flights they bought a ticket from. The chosen seat also indicates the flight class.
* **Choose Meal:** The customers can choose their meals from the available meal options for the flights they bought a ticket from and make the necessary payment.
* **Use Promotion:** The customers can benefit from the given promotions for the flights they bought a ticket from. The promotions can be used to earn free meals, gift miles, advantageous membership opportunities, or free lounge services.
* **Get Sale:** The customers can benefit from sales for the flights they bought a ticket from. The sales will allow customers to pay less for the ticket.

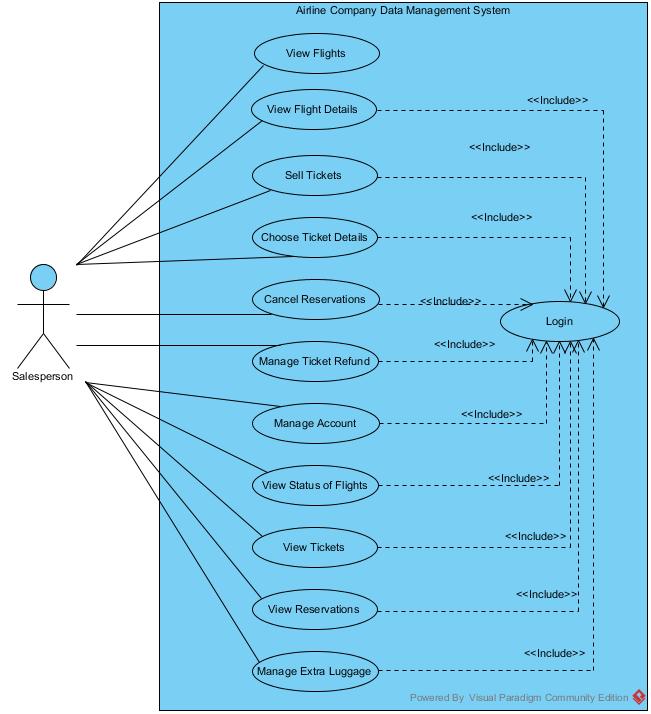
### Manager Use Cases



* **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. 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 details of the plane type such as size, capacity, seat map, model and delete the existing ones. Managers can also make changes in plane details such as passenger capacity, storage capacity, maximum flight time. 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. 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, update meal and class options.
* **View Status of Flights:** The manager can view status of all existing flights including their delay amount, class options and available meals.
* **Manage Promotion/Sales:** The manager can view all promotions and sales that exist in the system and for whom they can be applied. The campaign type and the sale amount and period are available for manager to see. He/she can add new sales or promotions according to the history of customers or flights by specifying the details of the promotions/sales.
* **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.
* **Manage prices:** The manager is able view the prices of all flights available in the system. He/she can assign prices to newly created flights and determine the price ranges according to airports, dates and times. The manager is also able to alter the prices of flights.
* **View Reservations:** The manager is able to view all current reservation and view the associated customer, date, route, plane, and, class details.
* **View Tickets:** The manager is able to view all current tickets and view the associated customer, date, route, plane, crew, meal, seat, luggage and, class details.
* **Manage Account:** The manager is able to view the details of his account, see username, password details. He/she can also update his account, change username/password, phone details.

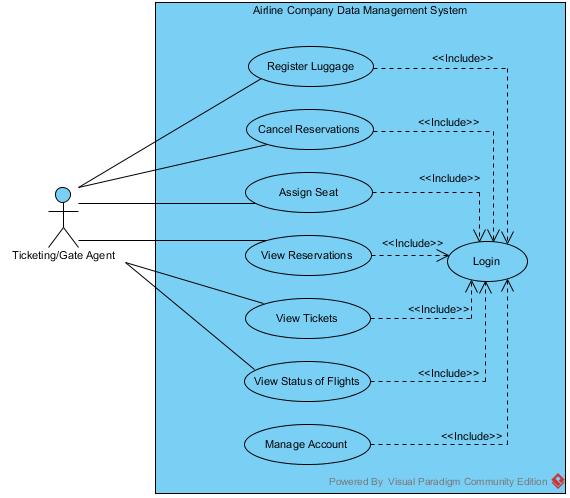
### Salesperson Use Cases

z



* **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.
* **View Flights:** The salesperson can view all flights that are registered to system. The user can specify date, arrival city and airport, and departure city and airport in order to help the customers to make reservations.
* **View Flight Details:** The salesperson can view flight details such as classes available, meals available, seat plan, and price. The salesperson is required to login to the system in order to be able to view flight details.
* **Sell Ticket:** The salesperson can complete the ticket purchasing operations for the customers. From the customer reservations, the salesperson can complete the purchasing operations using the credit card information of the customer.
* **Choose Ticket Details:** The salesperson can select the ticket details such as meal, class, seat number for the customer. He/she can also make the necessary payments 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.
* **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 Status of Flights:** 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 current reservation and view the associated customer, date, route, plane, and, class details.
* **View Tickets:** The salesperson is able to view all current tickets 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.

### Ticketing/Gate Agent Use Cases



* **Login:** The ticketing/gate agent can login to the system with his/her username and password. The ticketing/gate agents will be given access to Ticketing/gate agents functions when the account details are approved.
* **Assign Seat:** The ticketing/gate agent can assign seat for the ticket if the seat is not assigned yet. The seat is selected from the available seats.
* **Cancel Reservation:** The ticketing/gate agent can cancel customer reservations when necessary by deleting the reservation from the system.
* **View Status of Flights:** The ticketing/gate agent can view the status of all existing flights including their delay amount, class options and available meals.
* **View Reservations:** The ticketing/gate agent is able to view all current reservation and view the associated customer, date, route, plane, and, class details.
* **View Tickets:** The ticketing/gate agent is able to view all current tickets and view the associated customer, date, route, plane, crew, meal, seat, luggage and, class details.
* **Manage Account:** The ticketing/gate agent 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.
* **Register Luggage:** The ticketing/gate agent can register the luggage of the customers to the system.

## Algorithms

### Ticket 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 memberships, 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.

Ticket’s other attributes will affect the price as well. Ticket class, extra\_luggage and meal will be considered when setting a default price for a ticket.

### Promotion Related Algorithms

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

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

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

Beside these base promotions, Managers can set a promotion to a customer with customer’s user\_name so that the customer who has lots of cancelled reservations can be given less amount of promotions. For every 3 cancellations one future or available sale or promotion is cancelled.

### Reservation Related Algorithms

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

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

## Data Structures

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

# USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

# ADVANCED DATABASE COMPONENTS

## Views

### Customer Flight View

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

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

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

FROM flight NATURAL JOIN route )

### Customer Reservation View

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

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

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

FROM reservation NATURAL JOIN flight NATURAL JOIN route

WHERE user\_name = @userName)

### Customer Flight Details View

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

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

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

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

WHERE @userName in ((SELECT user\_name

FROM reservation R

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

(SELECT user\_name

FROM ticket T

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

and user\_name = @userName

### Customer Ticket View

The customer can only see the ticket details such as flight id, flight time, date, cities, luggage, seat\_no etc. The plane and 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, penalty\_amount, seat\_no, price) AS

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

FROM ticket NATURAL JOIN flight NATURAL JOIN route

WHERE user\_name = @userName)

### Manager Customer View

The manager cannot access to password and credit\_card\_details of the customers. Other customer information can be accessed by the manager.

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

(SELECT user\_name, name, birthdate, passport\_no, mile\_sum

FROM customer )

### Salesperson/Agent Flight Details View

The salesperson and ticketing/gate agent can see details of all flights. However, 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 SalespersonAgentFlightDetailsView(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 )

## Stored Procedures

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

### Make Reservation Stored Procedure

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

### Cancel Reservation Stored Procedure

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

### Purchase Ticket Stored Procedure

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

### Refund Ticket Stored Procedure

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

### Delete Airport Procedure

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

### Delete Route Procedure

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

### Cancel Flight Procedure

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

### Send Plane to Repair Stored Procedure

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

## Reports

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

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

WITH allReservationsAndTickets( reservationCount, saleCount ) AS

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

FROM reservation R, ticket T)

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

FROM allReservationsAndTickets, cutomer C

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

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

WITH totalMoney( reservationCount, saleCount ) AS

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

FROM reservation R, ticket T)

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

FROM Flight, totalMoney, Ticket

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

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

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

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

FROM Pilot )

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

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

FROM FlightAttendance)

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

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

FROM Manager )

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

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

FROM salesperson)

agents (agent\_count, agent\_avg\_sal) AS

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

FROM Ticketing/GateAgent)

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

FROM pilots, flightattendance, managers, salepersons, agents

### Total Number of Flights Associated with Each Airport, the Cities of the Airports, and the Number of Customers Using Each Airport

Calculates the total number flights arriving from/departing at each airport, the cities of the airports and the number of customers arriving at each airport.

SELECT count(flight), airport, sum(capacity)

FROM flight, airport, plane, route, plane type

WHERE flight.route\_id = route.route\_id and

flight.plane\_type = plane.plane\_type and

plane.plane\_type\_id = plane\_type.plane\_type\_id and

(route.departs = airport.city\_name or route.arrives = airport.city\_name)

GROUP BY airport.city\_name

## Triggers

* On insert operation to Manager, Salesperson, or Ticketing/Gate Agent tables, corresponding rows will be inserted to Reservation Authoritative and Staff tables.
* On insert operation to Pilot or Flight Attendance, corresponding rows will be inserted to Crew and Staff tables.
* On insert operation to Flight, if there is no plane with the given plane\_name, the plane will be added to planes.
* On insert operation to Airport, if there is no airport with the given city\_name, corresponding row will be added to City.
* On update operation for cancellation of a reservation, mile\_sum of the customer will be decreased as much as the taken distance point from that specific reservation.
* On delete operation to a ticket, mile\_sum of the customer will be decreased with additional penalty for this customer.
* On insert operation to reservation, distance of the flight will be added to mile\_sum of the customer.
* On delete operation on ticket, the corresponding row will be updated as cancelled from reservations.

## Constraints

There are limitations to the database system so that it is realistic and stable as possible:

* In order to interact with the system, users must be signed in. Only the available flights can be seen without login operation.
* Reservation Authoritative and Customers access the system with their unique user\_name and their very own password. They cannot have conflicting information i.e. user\_name.
* Customers can only use the promotions that are belonging to them. Promotions cannot be combined with other customers or promotions.
* Promotions cannot be transferred to any other customer.
* After not buying a reserved ticket in three days, the customer will not be able to have the same price for the flight.
* Prices for the tickets will raise regarding the time left for the flight.
* Plane assigned to a flight will be changed depending on the density at that route. Planes with the highest capacities will be assigned to overseas flights.
* A route, flight or plane cannot be removed from the system if there exist a ticket bought or crew members assigned to work in them.
* In round trip flights, returning date cannot be earlier than departure date.
* A customer cannot buy multiple tickets for the same exact time periods.
* Customers cannot have more than two extra luggage assigned to a ticket.
* Crew members cannot serve more than they are limited to.
* Dates and departure times of flights cannot conflict any other flights around that time.
* Planes cannot have more passengers than their capacities.
* Pilots are not able to be on duty more than their max\_flight\_distances.
* Business class tickets cannot be anywhere else front rows.
* Customers with insufficient mile\_sum cannot be given promotions.
* sale\_period in Sale table must be in the length of specified time. The period cannot be zero or over the time.
* The crew cannot be assigned to flights associated with airport cities other than the current\_location of the crew.
* No less than two pilots can be assigned to a flight.
* The total luggage amount associated with a flight cannot exceed the storage\_capacity of the assigned plane.
* The flight\_time of a route cannot exceed the max\_flight\_time of the plane assigned to flight.
* No ticket information can exist for a cancelled reservation.