DBMS: Airline Management

Topic: Airline Management System

<u>**ASaP Airlines**</u>

Members:

Arko Sharma (111601002)

S Muhammad Mustafa Sharief(111601022)

Prabhjot Singh(111601016)

(Team G.)

========================= TABLE OF CONTENTS============================

Topic: Airline Management System **1**

ASP Airlines 1

Introduction 3

Implementation and Dependencies 3

Client Side Requirements 3

Server Side Functionalities 4

ER Diagram 5

Relations 5

Views, Roles and Triggers 6

Use Cases 9

Normalization 14

Important Functions and Procedures 15

GUI Details 18

Introduction

This project is an endeavour to explore the paradigms of database management systems and their utility in today's world. Choosing a consequential and intricate topic of airline management systems, we have tried to simulate a fully functioning real - time , user facing application in order to understand the design principles required to craft a system appropriate for a particular set of requirements. In addition to this, we have also explored the customer facing requirements of the design by delving into website development in order to make all the functionalities available smoothly to an end user while keeping efficiency and scale in mind. This project is essentially a set of layers of design principles and implementation efforts, stacked progressively on top of each other in a way that gradually enriched our domain knowledge as well as made us familiar with implementation challenges intrinsic of creating any large project.

Implementation and Dependencies

a. Design a common web server as the back - end, which will contain all the necessary information required to provide, modify and review airline functionalities.
b. Design a web interface for client-side users providing them the option to search for and select flights between desired destinations.
c. Design a web interface for server-side users to alter the airline services.

Client Side Requirements

a. Searching for flights :

   The user gives departure location and day, destination location and day and we give a list of flights sorted in a particular order.

   Requirements: 2 lists containing choices that match available source and destination flights and an interface to select the desired flights.

b. Choice of ordering such as price, time of flight, etc.

   Requirements: 1 list describing suitably chosen orders and default order and functionality of refreshing search after changing the order of preference.

c. Choice of discounts based on occupation, age, etc.

   Requirements: Checklist based discount selection interface; displaying the modified cost of travel.

d. Option of booking a flight by entering the name and email address.

   Every reservation needs to have an associated passenger.

   Requirements: Entering name and email, storing them efficiently and associating them efficiently with the selected flight.

e. Option of checking in, ie, upon entering the PNR, printing e-boarding pass.

   Requirements: An interface for displaying the e-boarding pass, the ability to query the database using PNR efficiently and relaying the results to the front end.

Finally, after selecting all the options (or a few options and other default options) there needs to be efficient querying of the underlying database to search and display the sorted flights.

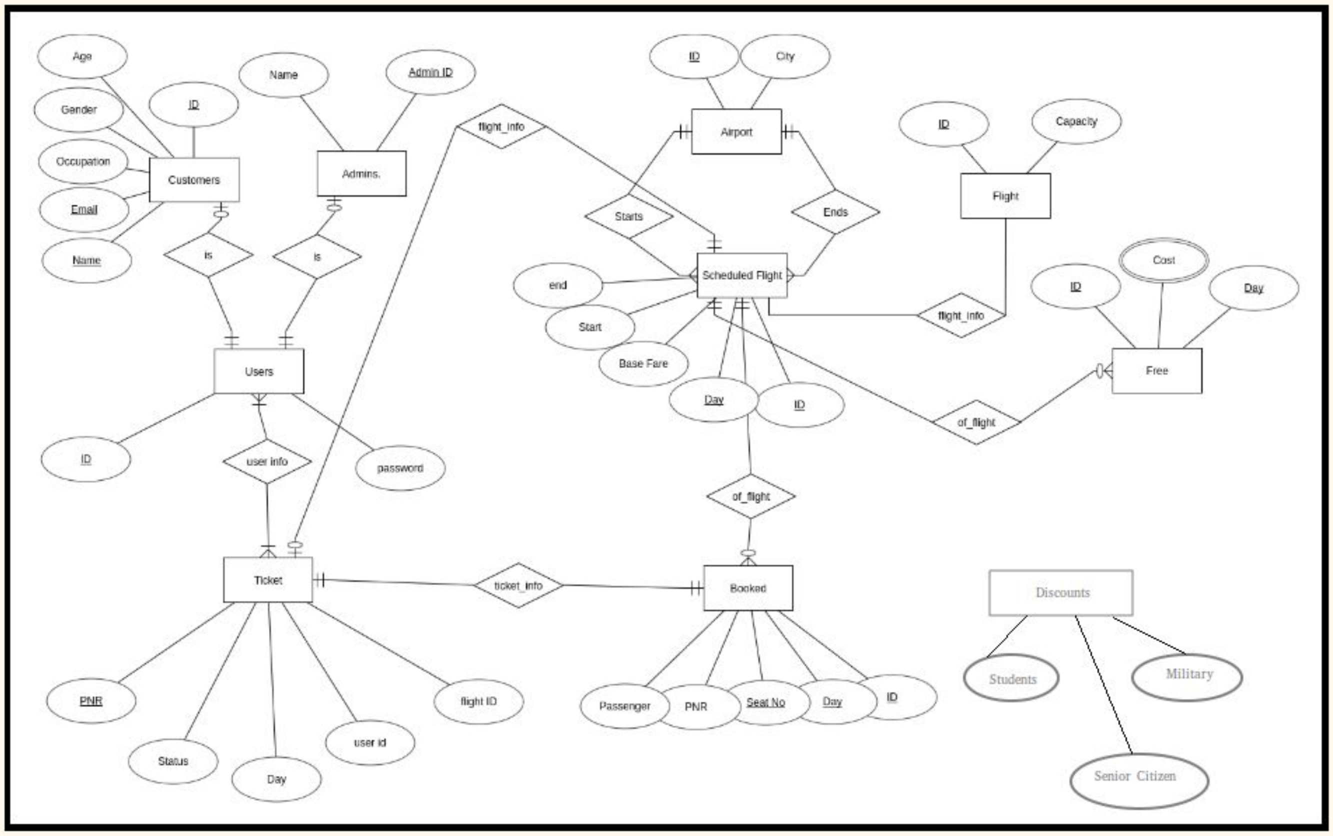
Server Side Functionalities

a. Adding / Deleting flights .
b. Adding /Deleting airports.
c. Adding / Deleting choice of discount.
d. Looking into flight details.
e. Checking flight status.

Requirements: All the server side requirements firstly require an easy user interface to select the desired action and the interface needs to link with the database to alter the records according to the option selected by the flight admin. The change should be such that it reflects itself in all associated relationships (integrity) and should also be efficiently implemented.

ER Diagram

Contribution : The design was the brainchild of active discussions by all the members. Each table was discussed and put through after its idea being scrutinized by all the members as to how it would be used and what its significance would be in the database.



Relations

1. Flight -> This table stores the attributes of all flights that are present. ID is the primary key and the only other attribute is the capacity of the flight.
2. Scheduled Flights -> To reduce redundancy and introduce modularity, we separate the list of flights currently scheduled from the list of all available flights in a separate table. This table is used whenever we need to search for flights between a given source and a given destination, so this table actually stores the routes of all scheduled flights.
3. Booked -> This table maps the booked seats to the actual passengers they have been reserved for. This table is also used to maintain a count of the number of seats that have been booked for each flight and to check the integrity constraint that the number of booked seats cannot exceed capacity.
4. Free -> This table is used to allow for dynamic pricing of flight seats. Many - a - times, companies require to increase or decrease the prices to incentivize customers and so, this table stores the actual price of the flights which can be updated as required.
5. User -> This table is used to manage the clients of the application.

6. Customers -> This table stores the user information for people who want to book flights.
7. Admins -> This table stores user information for registered admins who can view and modify flight details.
8. Ticket -> A table storing the details against each booked ticket. This table is queried when users want to review their flight details by entering the PNR.
9. Discount -> A list of available discount options used to calculate the sale price of tickets based on the choice selected by the user booking the flights. This table is updated whenever

Views, Roles and Triggers

1. Roles and Views

There are two overall types of users of the DBMS: those who book the flights(passengers) and those who add or remove flights and airports (flight - admins).

Hence we have 2 types of roles :

A. *Flight Admin Role* - Having all privileges on all tables.
B. *Passenger Role* - Having only select privilege on the scheduled_flight, airport, discount and ticket tables.

For admins, all attributes are visible .

For passengers we do not allow the entire tables but only the following views :
A. Airport - View containing only city and not ID.

B. Flight - View containing :

ID, Day from *scheduled_flights*,

Cost from *free* table - in order to know the cost before booking

All attributes from *tickets* table to access info about the booked ticket.

2. Triggers

Following are a set of obvious triggers to maintain consistency :

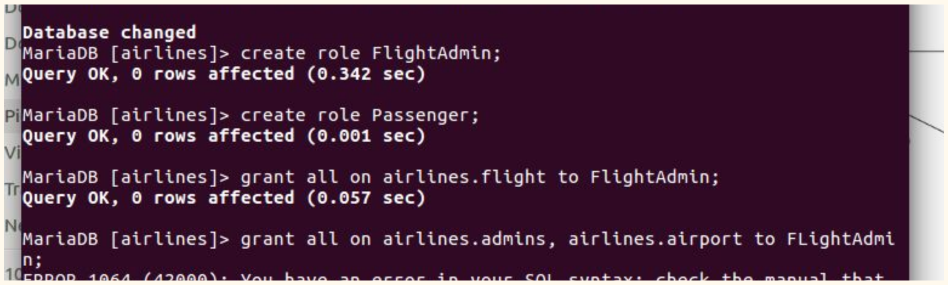
A. Before deleting an airport it should be made sure that no flight starts or ends from that airport or else it should be rolled back (actually enforced using foreign key).
B. Before inserting a seat in the booked flight(basically booking a ticket), we should check if the number of booked seats of that particular flight on that particular day is less than the capacity of the flight.
C. Before updating the real cost of the flight, we should make sure that the new cost must be greater than or equal to the base cost of the corresponding scheduled flight.
D. For maintaining consistency and monitoring updates , some other triggers can also be added.

Views, Roles and Triggers Implementation :

Contribution :

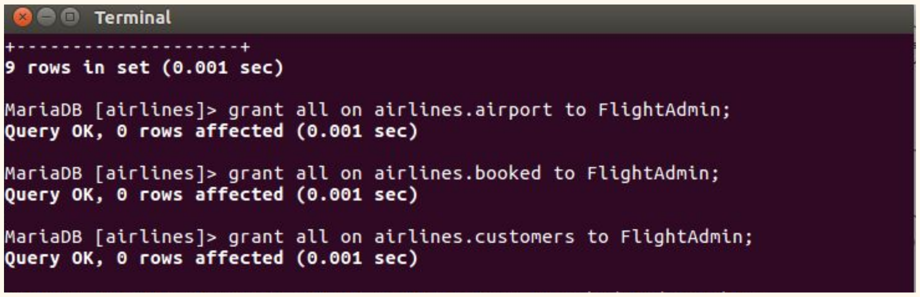
Arko - Created roles, views and privileges.

Sharief and Prabhjot - Trigger Implementations (3 in this section and 1 in the procedure section).



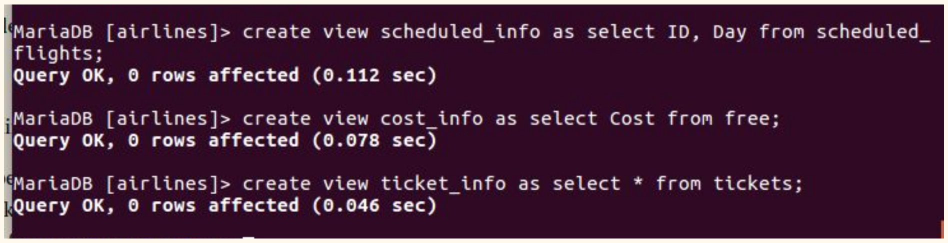
```
Database changed
MariaDB [airlines]> create role FlightAdmin;
Query OK, 0 rows affected (0.342 sec)

MariaDB [airlines]> create role Passenger;
Query OK, 0 rows affected (0.001 sec)

MariaDB [airlines]> grant all on airlines.flight to FlightAdmin;
Query OK, 0 rows affected (0.057 sec)

MariaDB [airlines]> grant all on airlines.admins, airlines.airport to FLightAdmi
n;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
```

Flight Admin and Passenger Roles.



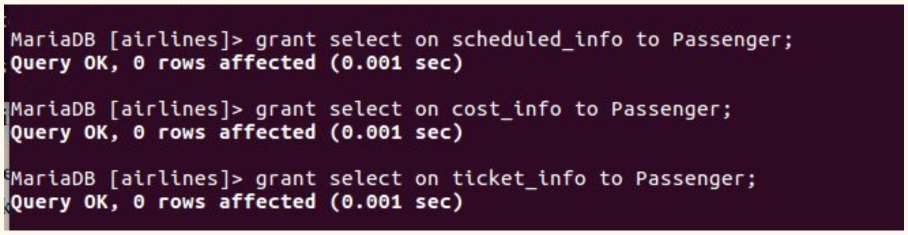
```
+--------------------+
9 rows in set (0.001 sec)

MariaDB [airlines]> grant all on airlines.airport to FlightAdmin;
Query OK, 0 rows affected (0.001 sec)

MariaDB [airlines]> grant all on airlines.booked to FlightAdmin;
Query OK, 0 rows affected (0.001 sec)

MariaDB [airlines]> grant all on airlines.customers to FlightAdmin;
Query OK, 0 rows affected (0.001 sec)
```

Giving all privileges to Flight admin on all tables.



```
MariaDB [airlines]> create view scheduled_info as select ID, Day from scheduled_
flights;
Query OK, 0 rows affected (0.112 sec)

MariaDB [airlines]> create view cost_info as select Cost from free;
Query OK, 0 rows affected (0.078 sec)

MariaDB [airlines]> create view ticket_info as select * from tickets;
Query OK, 0 rows affected (0.046 sec)
```

Creating views accessible to passengers.

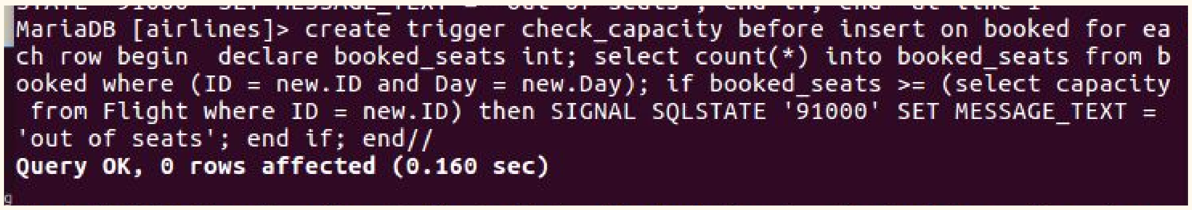


```
MariaDB [airlines]> grant select on scheduled_info to Passenger;
Query OK, 0 rows affected (0.001 sec)

MariaDB [airlines]> grant select on cost_info to Passenger;
Query OK, 0 rows affected (0.001 sec)

MariaDB [airlines]> grant select on ticket_info to Passenger;
Query OK, 0 rows affected (0.001 sec)
```

Granting privileges to passengers.

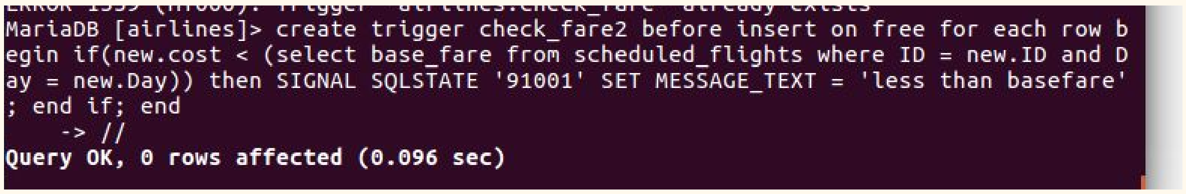
Trigger Implementations :

Trigger A implemented using Foreign Key constraint instead.



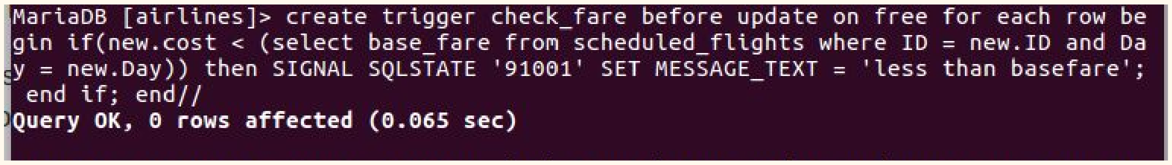
```
MariaDB [airlines]> create trigger check_capacity before insert on booked for ea
ch row begin  declare booked_seats int; select count(*) into booked_seats from b
ooked where (ID = new.ID and Day = new.Day); if booked_seats >= (select capacity
 from Flight where ID = new.ID) then SIGNAL SQLSTATE '91000' SET MESSAGE_TEXT =
'out of seats'; end if; end//
Query OK, 0 rows affected (0.160 sec)
```

Trigger B.



```
MariaDB [airlines]> create trigger check_fare2 before insert on free for each row b
egin if(new.cost < (select base_fare from scheduled_flights where ID = new.ID and D
ay = new.Day)) then SIGNAL SQLSTATE '91001' SET MESSAGE_TEXT = 'less than basefare'
; end if; end
    -> //
Query OK, 0 rows affected (0.096 sec)
```

Trigger C for insert.



```
MariaDB [airlines]> create trigger check_fare before update on free for each row be
gin if(new.cost < (select base_fare from scheduled_flights where ID = new.ID and Da
y = new.Day)) then SIGNAL SQLSTATE '91001' SET MESSAGE_TEXT = 'less than basefare';
 end if; end//
Query OK, 0 rows affected (0.065 sec)
```

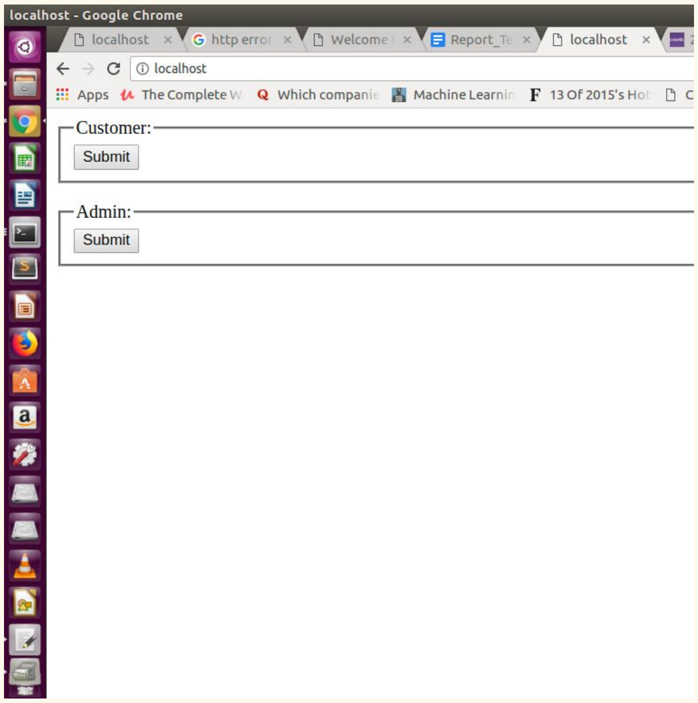
Trigger C for update.

In addition to these, there’s another trigger used while calling procedures and has been discussed later.

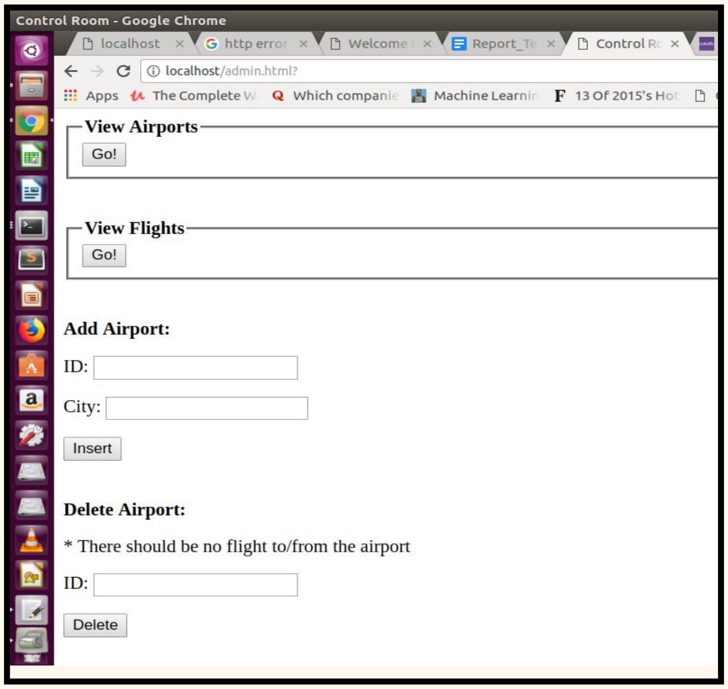
Use Cases

A few test cases that were displayed in the lab-session.

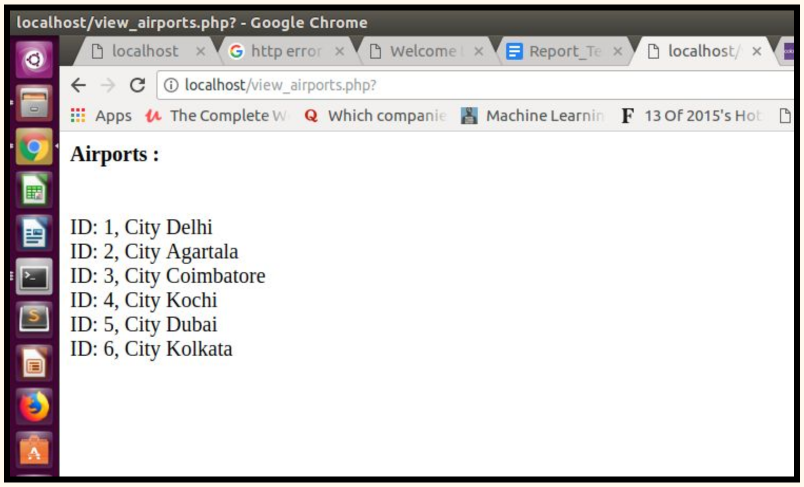
1. Login Page : Selecting type of user:



2. Admin level Functionalities : Adding / deleting airports, checking flight status.

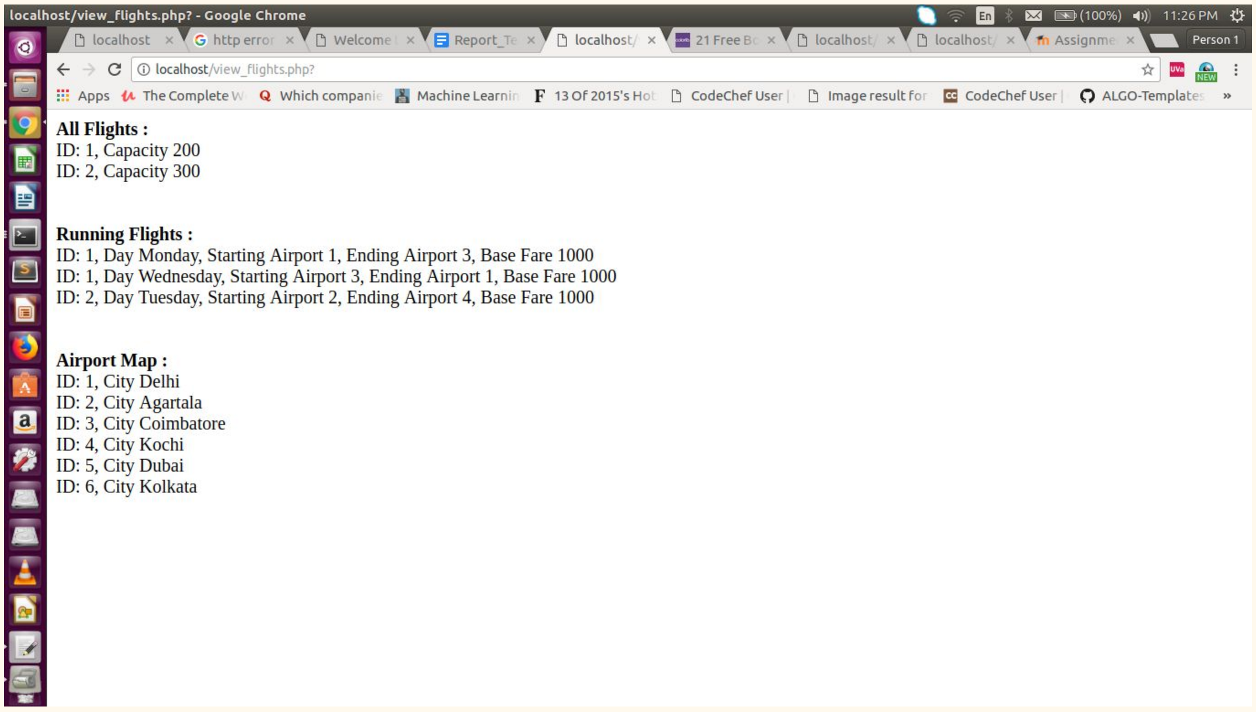


Admin - interface.


localhost/view_airports.php? - Google Chrome

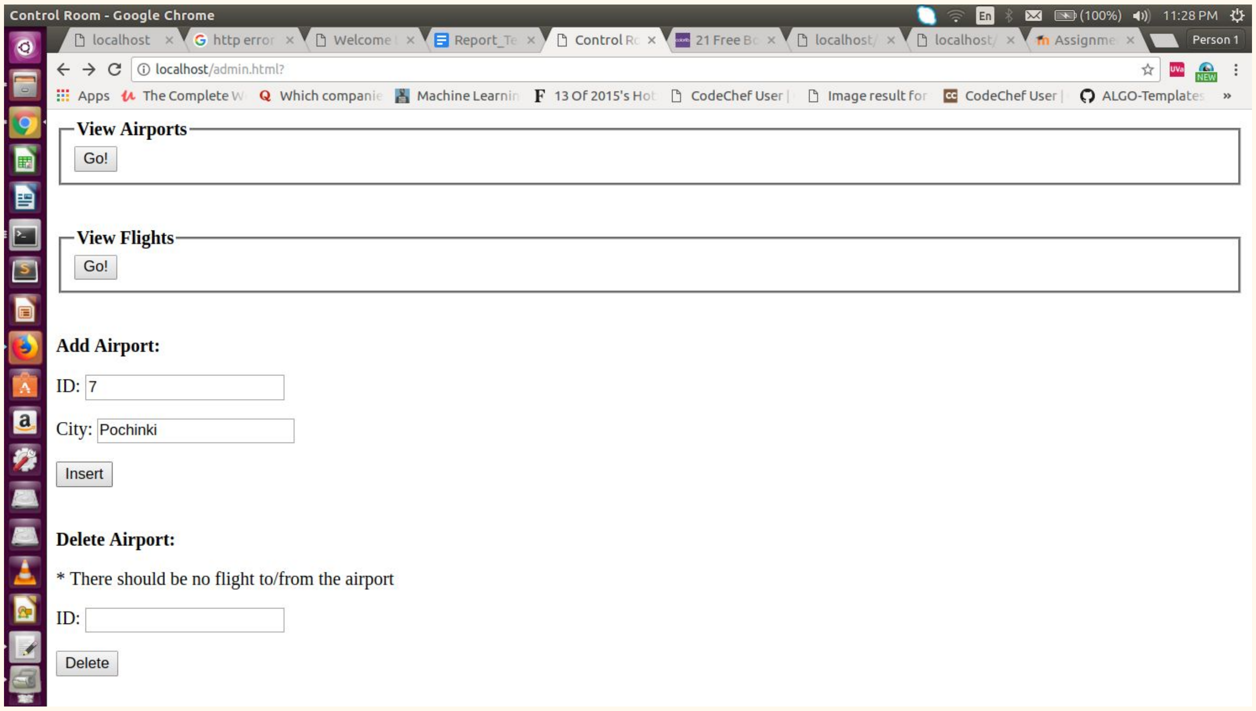
```
Airports :

ID: 1, City Delhi
ID: 2, City Agartala
ID: 3, City Coimbatore
ID: 4, City Kochi
ID: 5, City Dubai
ID: 6, City Kolkata
```

Viewing Airports.


localhost/view_flights.php? - Google Chrome

```
All Flights :
ID: 1, Capacity 200
ID: 2, Capacity 300

Running Flights :
ID: 1, Day Monday, Starting Airport 1, Ending Airport 3, Base Fare 1000
ID: 1, Day Wednesday, Starting Airport 3, Ending Airport 1, Base Fare 1000
ID: 2, Day Tuesday, Starting Airport 2, Ending Airport 4, Base Fare 1000

Airport Map :
ID: 1, City Delhi
ID: 2, City Agartala
ID: 3, City Coimbatore
ID: 4, City Kochi
ID: 5, City Dubai
ID: 6, City Kolkata
```

Viewing flights (Scheduled flights have “RUNNING” status).



View Airports

Go!

View Flights

Go!

Add Airport:

ID: 7

City: Pochinki

Insert

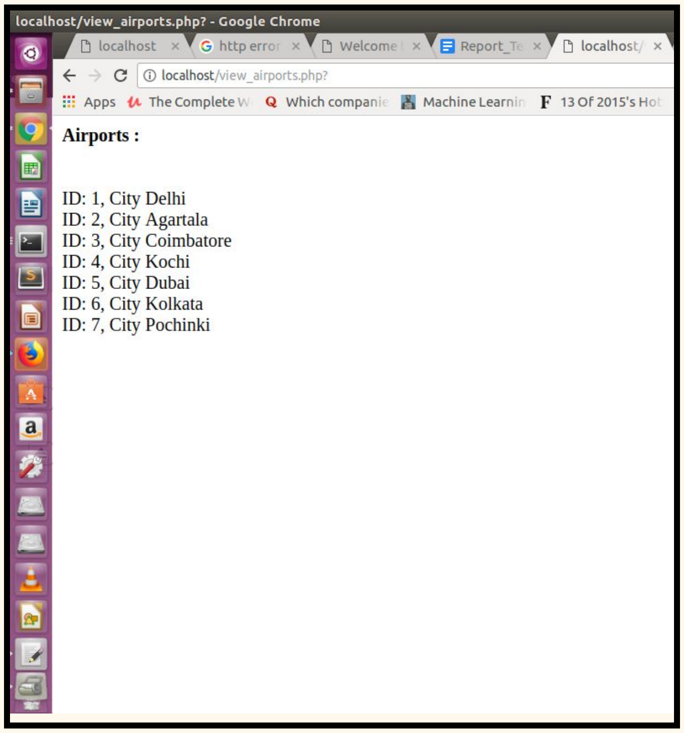
Delete Airport:

* There should be no flight to/from the airport

ID:

Delete

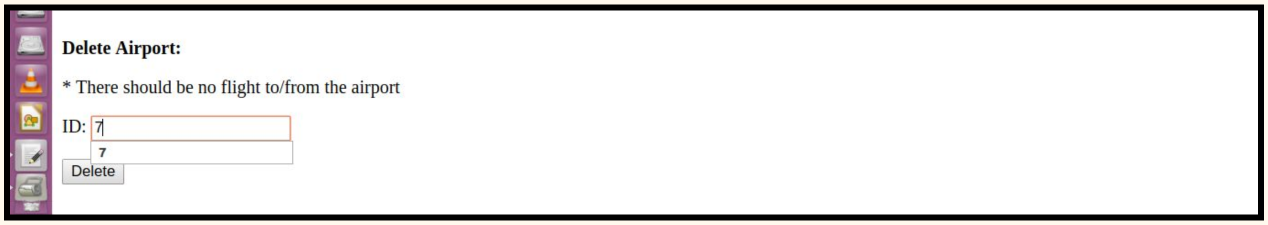
Inserting Pochinki airport.


localhost/view_airports.php? - Google Chrome

Airports :

ID: 1, City Delhi
ID: 2, City Agartala
ID: 3, City Coimbatore
ID: 4, City Kochi
ID: 5, City Dubai
ID: 6, City Kolkata
ID: 7, City Pochinki

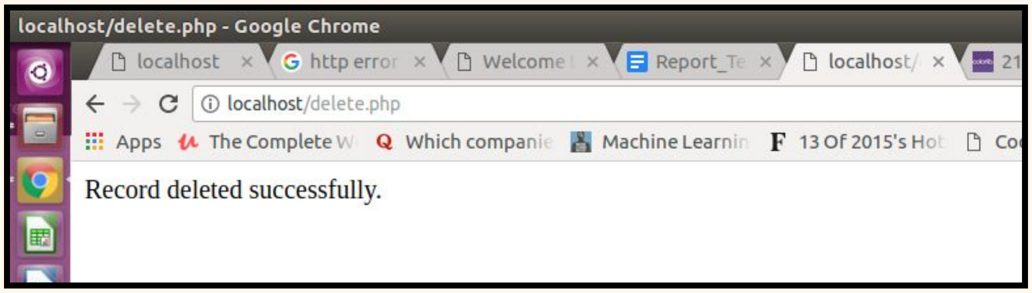
Showing airports after inserting.


Delete Airport:

* There should be no flight to/from the airport

ID: 7

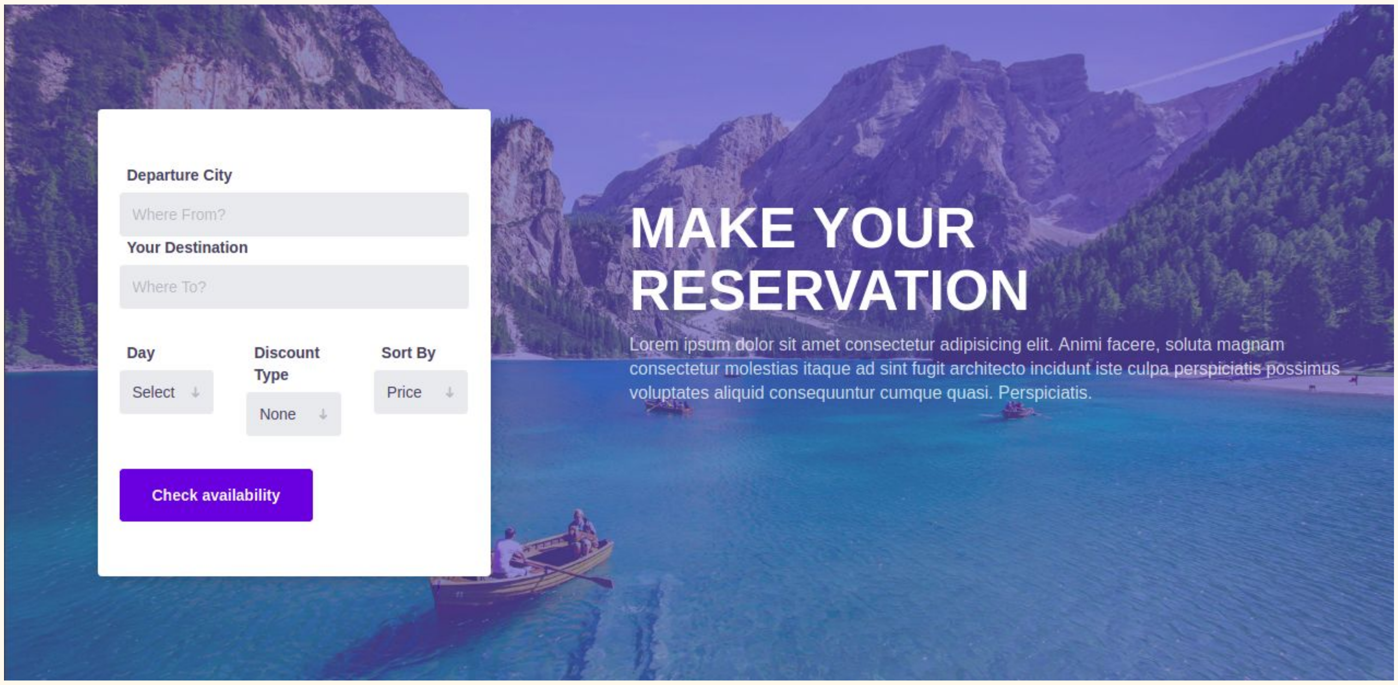
Deleting “Pochinki” airport.


localhost/delete.php - Google Chrome

Record deleted successfully.

TBA : Adding / Deleting flights , choice of discounts.

3. Customer level functionalities: Searching for flights.


Departure City
Where From?
Your Destination
Where To?
Day
Select
Discount Type
None
Sort By
Price
Check availability
MAKE YOUR RESERVATION
Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi facere, soluta magnam consectetur molestias itaque ad sint fugit architecto incidunt iste culpa perspiciatis possimus voluptates aliquid consequuntur cumque quasi. Perspiciatis.

Normalization

Contribution : Everyone checked each table to be convinced of the dependencies present.

To reduce redundancy and ensure storage efficiency and scalability, we scrutinize the functional dependencies present in our database. We analyse in what format data could be inserted and try to tune our design to meet the normalization requirements.

First, we check for BCNF :

The only dependencies in our database are of the form (Superkey -> Attributes) except "booked" table.

So, the tables are in BCNF form except "booked" table where the following dependency holds :

PNR -> ID

PNR is not a superkey in the “booked” table. This is because a customer may book several seats while booking a particular flight and all of these seats have to be stored using the same PNR.But, ID is a part of the candidate key and removing this will cause loss of dependencies in that table which cannot be recovered.

So we choose to not remove this and keep it in the resulting 3NF form, bearing the cost of redundancy to preserve dependencies.

Important Functions and Procedures

Contribution :

Everyone was involved in active discussion about design , implementation and future courses of action.

Specifics :

Sharief - Wrote procedure 4.

Prabhjot - Wrote procedure 3.

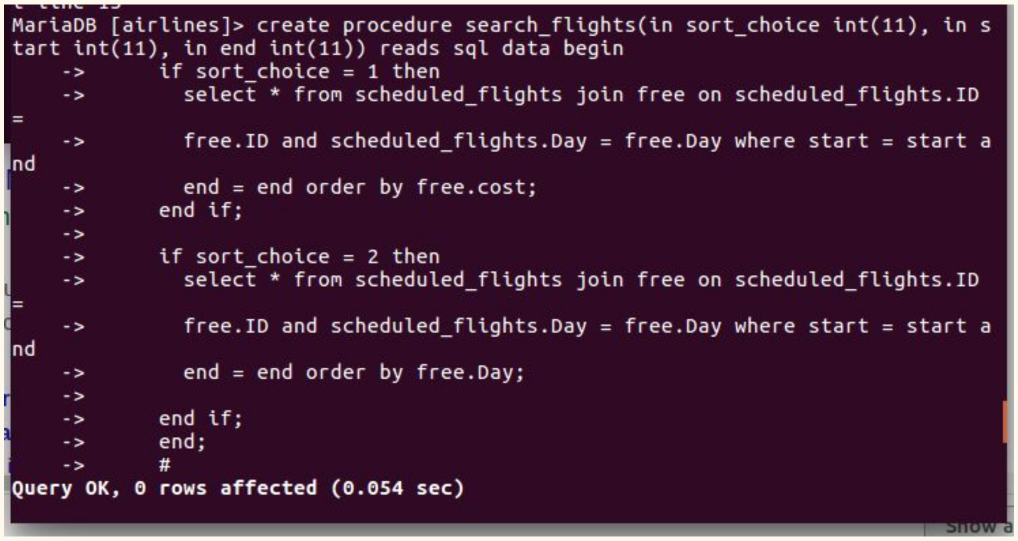
Arko - Wrote procedures 1,2.

---

1. A procedure to give a sorted list of flights from starting airport to ending airport. This procedure also provides the functionality of changing the sorting order - by price or by day of the flight.

   This is a critical procedure which is used to implement searching of flights in the GUI.

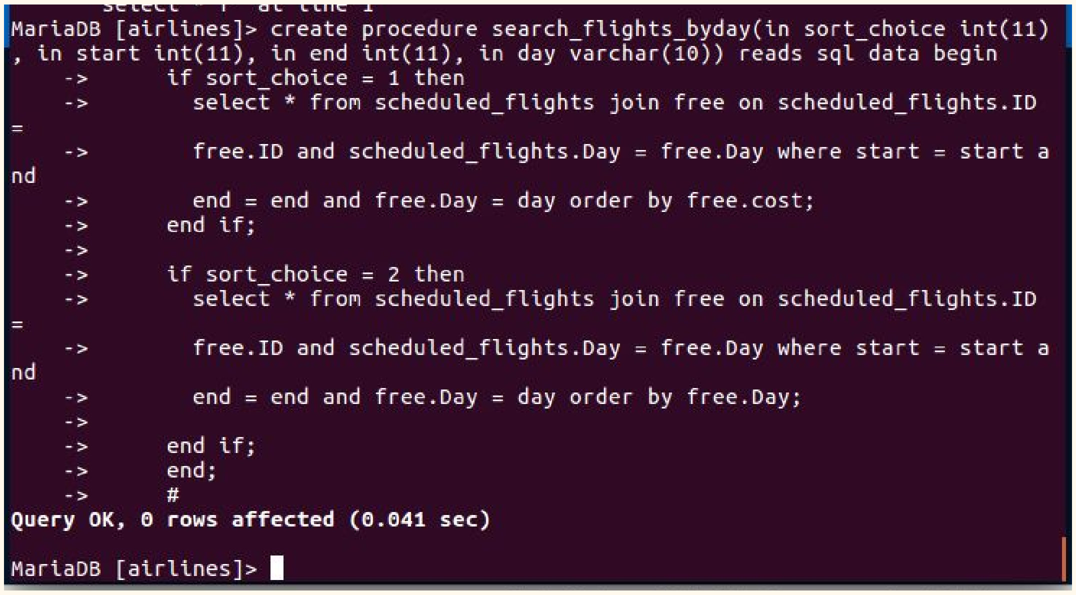
   The user may search for flights on a given day or for all days on for a given pair of source and destination. In the case when he chooses to view on all days, he has an option to sort the flights using the day of flight.



```
MariaDB [airlines]> create procedure search_flights(in sort_choice int(11), in s
tart int(11), in end int(11)) reads sql data begin
    ->      if sort_choice = 1 then
    ->        select * from scheduled_flights join free on scheduled_flights.ID
=
    ->        free.ID and scheduled_flights.Day = free.Day where start = start a
nd
    ->        end = end order by free.cost;
    ->      end if;
    ->
    ->      if sort_choice = 2 then
    ->        select * from scheduled_flights join free on scheduled_flights.ID
=
    ->        free.ID and scheduled_flights.Day = free.Day where start = start a
nd
    ->        end = end order by free.Day;
    ->
    ->      end if;
    ->      end;
    ->      #
Query OK, 0 rows affected (0.054 sec)
```

2. A procedure to give a sorted list of flights from a starting airport to another airport on a particular day.

   This procedure deals with the case where the user searches for flights by entering a particular value of day through the GUI.

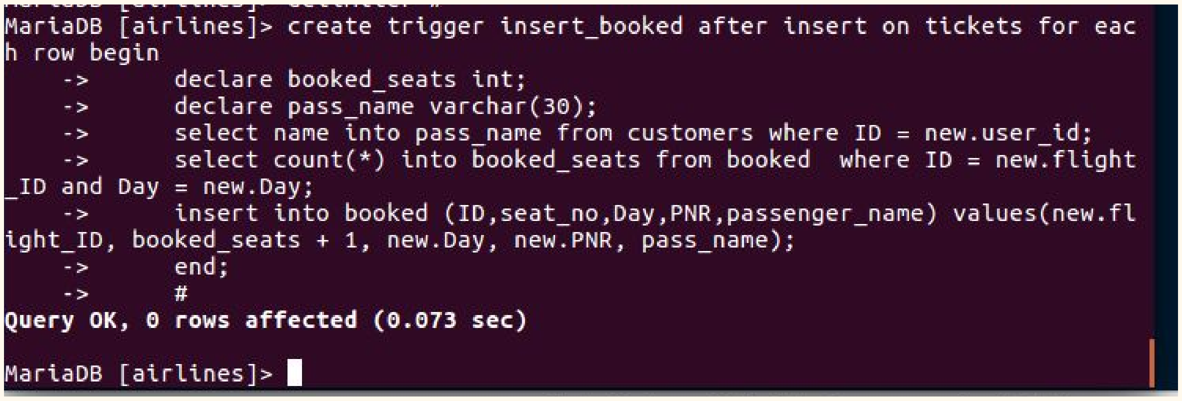


```
MariaDB [airlines]> create procedure search_flights_byday(in sort_choice int(11)
, in start int(11), in end int(11), in day varchar(10)) reads sql data begin
    ->      if sort_choice = 1 then
    ->        select * from scheduled_flights join free on scheduled_flights.ID
=
    ->        free.ID and scheduled_flights.Day = free.Day where start = start a
nd
    ->        end = end and free.Day = day order by free.cost;
    ->      end if;
    ->
    ->      if sort_choice = 2 then
    ->        select * from scheduled_flights join free on scheduled_flights.ID
=
    ->        free.ID and scheduled_flights.Day = free.Day where start = start a
nd
    ->        end = end and free.Day = day order by free.Day;
    ->
    ->      end if;
    ->      end;
    ->      #
Query OK, 0 rows affected (0.041 sec)

MariaDB [airlines]>
```

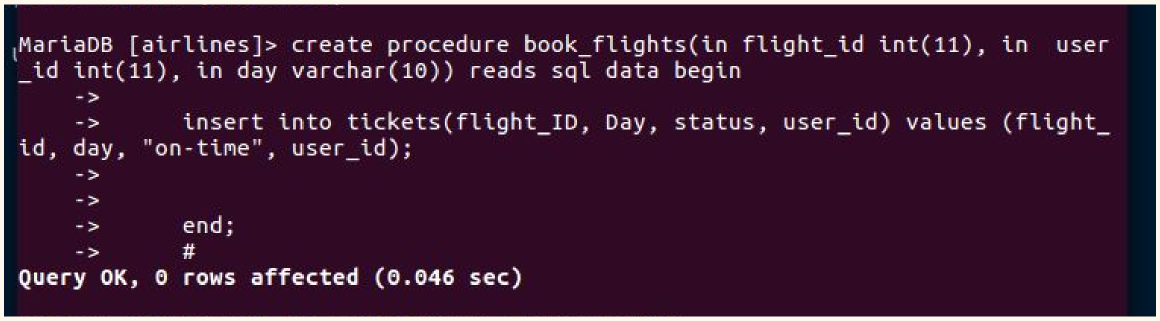
3. A procedure to implement booking of flights by inserting the particulars into the “tickets” table. PNR is auto generated through auto increment.

A trigger is used to insert the required values into the booked table. The newly generated PNR is used to update the booked table and the seat_no is generated by counting the number of seats of the flight previously booked .

This procedure implements the booking of a seat of a particular flight when the user enters his/her required details.



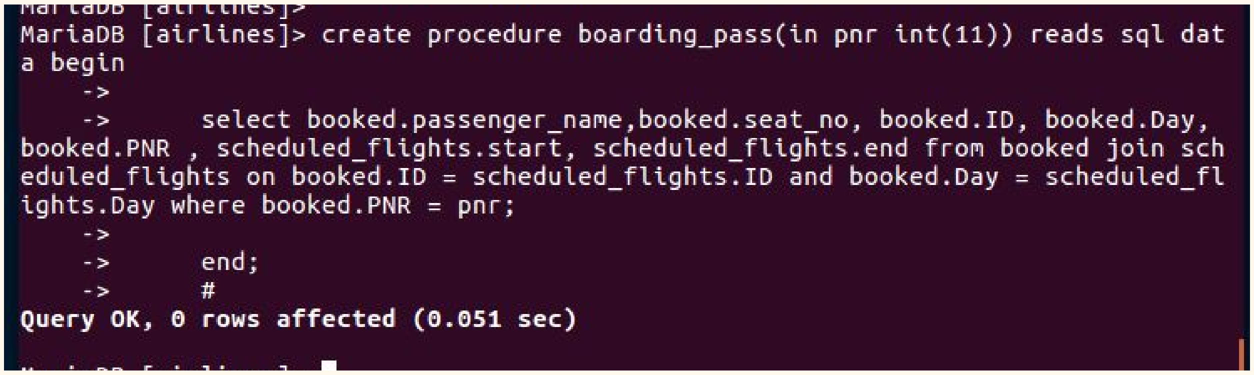
```
MariaDB [airlines]> create trigger insert_booked after insert on tickets for eac
h row begin
    ->      declare booked_seats int;
    ->      declare pass_name varchar(30);
    ->      select name into pass_name from customers where ID = new.user_id;
    ->      select count(*) into booked_seats from booked  where ID = new.flight
_ID and Day = new.Day;
    ->      insert into booked (ID,seat_no,Day,PNR,passenger_name) values(new.fl
ight_ID, booked_seats + 1, new.Day, new.PNR, pass_name);
    ->      end;
    ->      #
Query OK, 0 rows affected (0.073 sec)

MariaDB [airlines]>
```



```
MariaDB [airlines]> create procedure book_flights(in flight_id int(11), in  user
_id int(11), in day varchar(10)) reads sql data begin
    ->
    ->      insert into tickets(flight_ID, Day, status, user_id) values (flight_
id, day, "on-time", user_id);
    ->
    ->
    ->      end;
    ->      #
Query OK, 0 rows affected (0.046 sec)
```

4. A procedure to print the boarding pass given the PNR as the input.

We had a requirement to print the details of flights after booking using the PNR which is a unique identifier of booked tickets. This functionality is provided to both the flight admins as well as the passengers through the GUI. This is important to allow users to review the details of flights on demand and for admins to monitor and scrutinize tickets as required by checking if the PNR is valid or not.



```
MariaDB [airlines]> create procedure boarding_pass(in pnr int(11)) reads sql dat
a begin
    ->
    ->      select booked.passenger_name,booked.seat_no, booked.ID, booked.Day,
booked.PNR , scheduled_flights.start, scheduled_flights.end from booked join sch
eduled_flights on booked.ID = scheduled_flights.ID and booked.Day = scheduled_fl
ights.Day where booked.PNR = pnr;
    ->
    ->      end;
    ->      #
Query OK, 0 rows affected (0.051 sec)
```

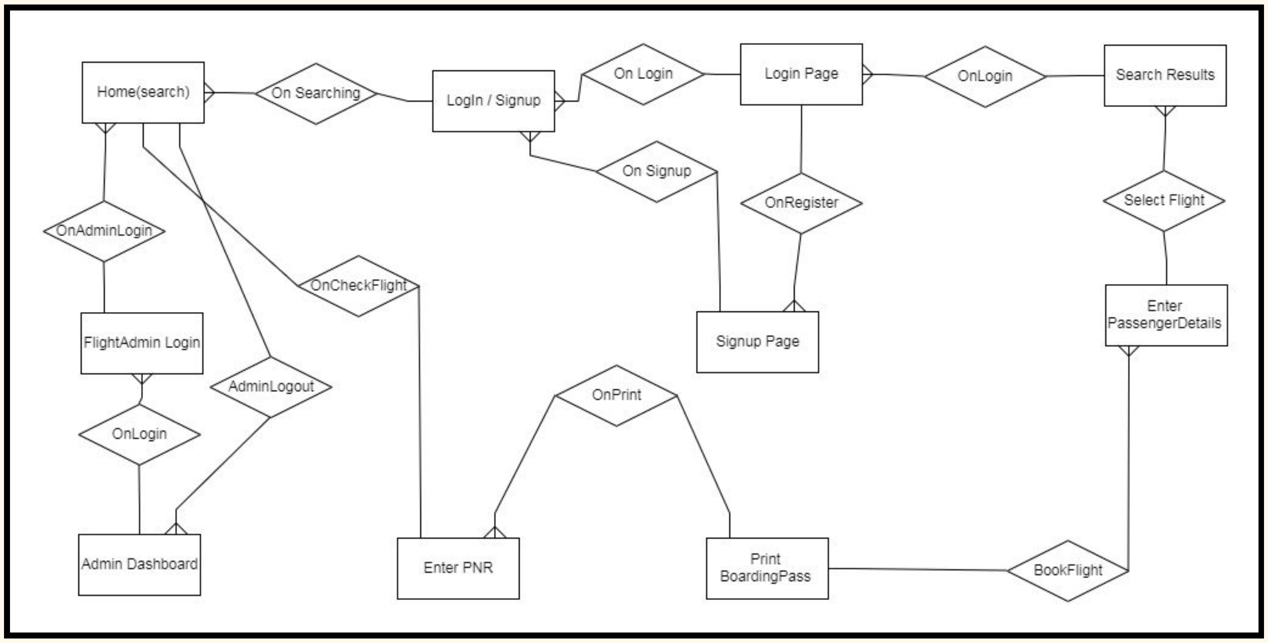
GUI Details

Contribution : Arko - Created the login/signup pages and underlying authentications.

Sharief - Created the admin dashboard and admin function pages.

Prabhjot - Created the customer-booking pages for searching and selecting flights.

* Design Layout*



1. The homepage shows the interface for searching for flights. This page redirects to a page having “login/signup” option and forwards successfully logged in users to the search results. Only registered customers have access to flight details of the application. This is accomplished by checking the log in details before forwarding to the results page.

2. Upon selecting “Sign-Up” option, the inbuilt user management system of the application is invoked which uses the user management of MariaDB. The resulting page asks for personal information such as name,password,age,occupation etc. During the transition to the resulting page, an admin inserts this information into the user and customer tables of the airlines application thereby creating a new customer (user) for the application. This admin is a MariaDB - admin and has the admin privileges of by mariadb. Even the checking of login details mentioned above is done by such an admin.

3. Now, if the login is successful (using the admin intervention as described above), then we display the search results to the customer. At this point, we only need to use the user - privileges that we defined above. For ensuring this we create a dummy user in mariadb whom we assign the *Passenger* role. Every time a customer logs in , this user is used to set up a connection to the database and carry on with the required actions. This is done to ensure that no passenger has access to admin privileges through the UI.
4. A separate page deals with requirements of the flight-admins. They are directed to a login page of themselves, where the login details are checked in the admin table instead of the customer table and if login is successful, redirected to the flight-admin dashboard to access required interfaces.
5. Finally a webpage is made available to print the details of booked tickets using an input of PNR by querying the *tickets* table.

Choices for improvements :

1. We can change the start/end on same days of the week but at different times.
2. Planning for returns of the flights
3. Giving different discounts for different companies (adding *companies* table to the design).