Airline Database Management

Airline industry is one of the largest and ever-growing businesses in the world. It caters to a vast majority of the population. These airlines are committed to provide a wonderful user experience. The aircrafts are equipped with modern day technologies which not only guarantee a safe flight, but also a comfortable journey to the passengers.

The Airlines offers certain kinds of discounts, based on the person- such as discounts for children and senior citizens and early booking of flights. Using a database, it becomes easier to keep track of the discounts.

Also, online reservations can be done for reservation of seats which are secure and allows customers the flexibility of booking seats from the convenience of their homes. Having a computerized database system to manage all the transactions of customers, along with keeping track of all the employees of the organization would be much easier.

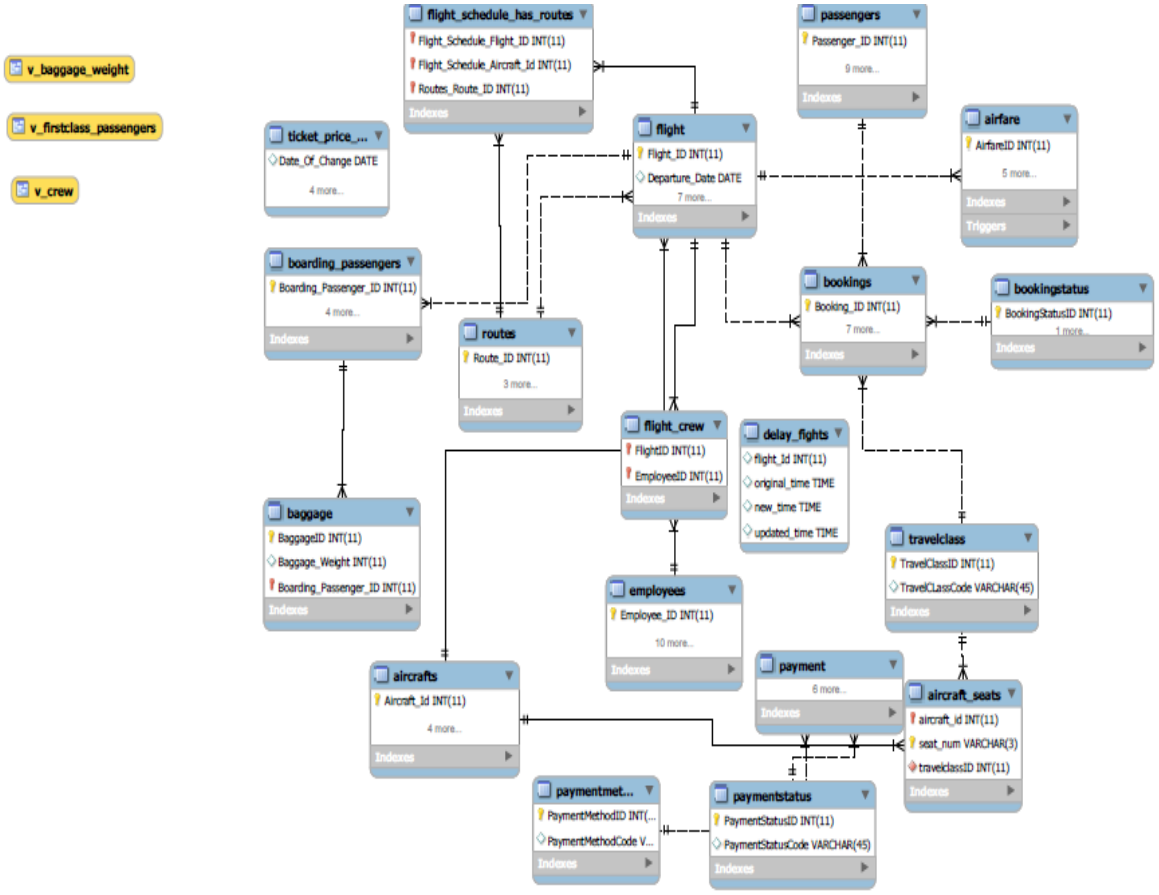
In this project I will be using stored procedures and transactions for adding customers or booking the seats on a flight. In case of cancellations, based on certain factors, the refund would be given to the customer.

Main Entities:

- **Aircrafts**: Contains the details of the aircrafts owned by the airline.
- **Customers**: Contains the details of the customers.
- **Employees**: Contains the details of the employees working for the airline.
- **Routes**: Each flight will have a route, based on the departure and arrival locations.
- **Flight Schedule**: This contains the date time, departure, arrival of each flight.
- **Booking**: Contains the transaction details such as booking a seat by customers.
- **Flight Crew**: Contains the discount details based on certain factors.

Relationships:

- There will be a one-to-one relationship between the flight schedule and aircrafts.
- There will be a many-to-many relationship between customers and transactions.
- There exists a many-to-many relationship between the flight schedule and routes.



An EER Diagram with connecting the entities and showing the relationships among them. Along with the views.

In this project I have used Stored Procedures, Triggers and Views which are described in detail below:

- Stored Procedures:
  - **sp_booking_seat:** This stored procedure is used to book a flight ticket for all the passengers. Here I have considered the date of booking and the date of departure of a particular flight, and based on that the passenger will be given a discount on the total fare. The screenshot is attached below.

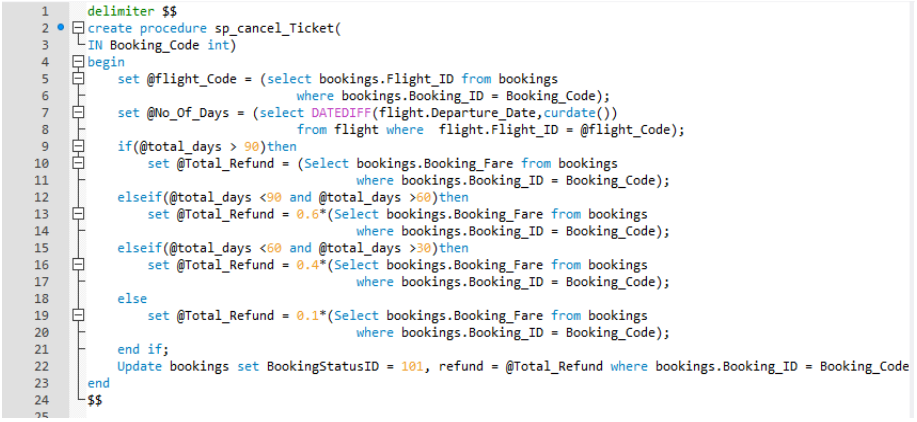
```
delimiter $$
create procedure sp_booking_seat(
IN passenger_ID int,
IN Flight_code int,
IN TravelClassID int,
IN booking_status_ID int)
begin

    if(TravelClassID = 1)  then
        set @Total_fare = (Select airfare.FC_Fare from airfare inner join flight
        on airfare.Flight_ID = flight.Flight_ID where flight.Flight_ID = Flight_code);
        set @total_days = (select DATEDIFF(flight.Departure_Date,curdate())
                           from flight where  Flight_ID = Flight_code);
                if(@total_days > 90)then
                    set @Total_fare = @Total_fare * 0.6;
                elseif(@total_days > 60)then
                    set @Total_fare = @Total_fare * 0.8;
                else
                    set @Total_fare = @Total_fare;
                end if;
        insert into bookings(Booking_Date,Flight_ID,BookingStatusID,TravelClassID,PassengerID,Booking_Fare)

    elseif(TravelClassID = 2)  then
        Set @Total_fare = (Select airfare.FC_Fare from airfare inner join flight
        on airfare.Flight_ID = flight.Flight_ID where flight.Flight_ID = Flight_code);
        set @total_days = (select DATEDIFF(flight.Departure_Date,curdate())
                           from flight where  flight.Flight_ID = Flight_code);
                if(@total_days > 90)then
                    set @Total_fare = @Total_fare * 0.6;
                elseif(@total_days > 60)then
                    set @Total_fare = @Total_fare * 0.8;
                else
                    set @Total_fare = @Total_fare;
                end if;
        insert into bookings (Booking_Date,Flight_ID,BookingStatusID,TravelClassID,PassengerID,Booking_Fare
    else
        Set @Total_fare = (Select airfare.FC_Fare from airfare inner join flight
        on airfare.Flight_ID = flight.Flight_ID where flight.Flight_ID = Flight_ID);
        set @total_days = (select DATEDIFF(flight.Departure_Date,curdate())
                           from flight where  flight.Flight_ID = Flight_code);
                if(@total_days > 90)then
                    set @Total_fare = @Total_fare * 0.6;
                elseif(@total_days > 60)then
                    set @Total_fare = @Total_fare * 0.8;
                else
                    set @Total_fare = @Total_fare;
                end if;
        insert into bookings (Booking_Date,Flight_ID,BookingStatusID,TravelClassID,PassengerID,Booking_Fare
    end if;
end$$

call sp_booking_seat(1013,505,1,100);
```

  - **sp_cancel_Ticket:** This stored procedure cancels a ticket booked by customer by providing a refund amount based on the number of days before departure the ticket was cancelled by the passenger.



```
delimiter $$
create procedure sp_cancel_Ticket(
IN Booking_Code int)
begin
    set @flight_Code = (select bookings.Flight_ID from bookings
                        where bookings.Booking_ID = Booking_Code);
    set @No_Of_Days = (select DATEDIFF(flight.Departure_Date,curdate())
                       from flight where  flight.Flight_ID = @flight_Code);
    if(@total_days > 90)then
        set @Total_Refund = (Select bookings.Booking_Fare from bookings
                             where bookings.Booking_ID = Booking_Code);
    elseif(@total_days <90 and @total_days >60)then
        set @Total_Refund = 0.6*(Select bookings.Booking_Fare from bookings
                             where bookings.Booking_ID = Booking_Code);
    elseif(@total_days <60 and @total_days >30)then
        set @Total_Refund = 0.4*(Select bookings.Booking_Fare from bookings
                             where bookings.Booking_ID = Booking_Code);
    else
        set @Total_Refund = 0.1*(Select bookings.Booking_Fare from bookings
                             where bookings.Booking_ID = Booking_Code);
    end if;
    Update bookings set BookingStatusID = 101, refund = @Total_Refund where bookings.Booking_ID = Booking_Code
end
$$
```

- **sp_alternate_flights:** This is a stored procedure to display an alternate list of flights for the same departure and arrival locations, when a flight is delayed or cancelled. This helps to re-allocate passengers from the delayed flight to another.

```
delimiter $$
create procedure sp_alternate_flights(
IN Flight_code int)
begin
    set @departure = (select  flight.Departure from flight where Flight_ID = Flight_code);
    set @arrival =   (select  flight.Arrival from flight where Flight_ID = Flight_code);

    select * from flight where flight.Departure = @departure and flight.Arrival = @arrival and
    flight.status <> 'Delayed';
end$$


call sp_alternate_flights(505);
```

- **sp_add_passengers:** This stored procedure is used to add a passenger details onto the database. In case a new passenger is to be added, this stored proc can be called directly by passing all the details. This can be used to control, who all have access to the database.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_add_passengers`(
IN First_Name varchar(50),
IN Last_Name varchar(50),
IN Date_Of_Birth date,
IN Address varchar(50),
IN City varchar(50),
IN State varchar(50),
IN Zipcode varchar(50),
IN Tel_No varchar(50),
IN Email varchar(50))
BEGIN
    insert into passengers (
    First_Name,
    Last_Name,
    Date_Of_Birth,
    Address, City,
    State, Zipcode,
    Tel_No, Email)
    values (First_Name ,Last_Name,Date_Of_Birth,Address,City,State,Zipcode,Tel_No,Email)
END
```

- **sp_layover_time:** This displays the layover time between the two flights which are in different routes.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_layover_time`(in route_1 INTEGER,in depa
in depart_ts_2 time, out layover time)
BEGIN
    set @arrive_time_1 = (SELECT Arrival_Time
     FROM flight
     WHERE RouteID = route_1
     AND Departure_Time = depart_ts_1);

     set @depart_time_2 = (SELECT Arrival_Time
     FROM flight
     WHERE RouteID = route_2
     AND Departure_Time = depart_ts_2);

set layover = (@depart_time_2 - @arrive_time_1);
END
```

- **Views:**
  - **v_baggage_weight:** This view is used to display all the passengers who have exceed the allowed baggage weight limit of 46 kilos. It is selected based on the passengers who are boarding the flights.

```
create view v_baggage_weight as
select passengers.Passenger_ID,boarding_passengers.FlightID,sum(baggage.Baggage_Weight)
from passengers inner join boarding_passengers on
passengers.Passenger_ID= boarding_passengers.PassengerID inner join baggage
on boarding_passengers.Boarding_Passenger_ID = baggage.Boarding_Passenger_ID
where boarding_passengers.Boarding_Passenger_ID in
(select Boarding_Passenger_ID
from baggage  group by Boarding_Passenger_ID
having sum(baggage.Baggage_Weight) > 46)
group by passengers.Passenger_ID,boarding_passengers.FlightID;

select * from v_baggage_weight;
```

  - **v_firstclass_passengers:** Generally, first class passengers are given preferential treatment over the other passengers. So, in case there is a delay in the departure of the flights or a flight is cancelled these passengers have to be accommodated elsewhere. This view will give us the details of those passengers whose flight was delayed.

```
create view v_firstclass_passengers as
select passengers.passenger_ID,passengers.First_Name,passengers.Last_Name,bookings.Flight_ID,
bookings.Booking_ID,bookingstatus.BookingStatusCode,travelclass.TravelCLassCode
from passengers inner join bookings on passengers.Passenger_ID = bookings.PassengerID
inner join bookingstatus on bookingstatus.BookingStatusID = bookings.BookingStatusID
inner join travelclass on travelclass.TravelClassID = bookings.TravelClassID
inner join flight on bookings.Flight_ID = flight.Flight_ID
where bookings.TravelClassID = 1 and flight.Status = 'Delayed';
```

  - **v_crew:** This gives us all the details of the crew members who are on a flight.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `v_crew` AS
    SELECT
        `employees`.`Employee_ID` AS `Employee_ID`,
        `employees`.`First_Name` AS `First_Name`,
        `employees`.`Last_Name` AS `Last_Name`,
        `employees`.`Role` AS `Role`,
        `flight`.`Flight_ID` AS `Flight_ID`
    FROM
        ((`employees`
        JOIN `flight_crew` ON ((`employees`.`Employee_ID` = `flight_crew`.`EmployeeID`)))
        JOIN `flight` ON ((`flight_crew`.`FlightID` = `flight`.`Flight_ID`)))
```

- **Trigger:**
  - **ticket_history:** A trigger which is fired when there is an update on the airfare. This trigger saves the changes of the price of the ticket in a separate table - **Ticket_Price_History** which can be referred in case of

any further updates.

```
delimiter $$;
create trigger ticket_history
before update on airfare
for each row
begin
insert into Ticket_Price_History values(curdate(),old.flight_ID,old.FC_Fare,old.BC_Fare,old.EC_Fare);
end;


update airfare set FC_Fare = 1005 where flight_ID = 506;
select * from Ticket_Price_History;
```

- **tr_update_payment:** A trigger is fired whenever a ticket is booked, and the values are inserted to the bookings table. This trigger updates the table payments, which contains the payment details.

```
delimiter $$
create trigger tr_update_payment after insert on bookings
for each row
begin
insert into payment(PaymentDate,PaymentAmount,PaymentMethodID,PaymentStatusID,Booking_ID)
values(curdate(),new.Booking_Fare,1,4,new.Booking_ID);
end
$$
```

```
select count(payment.PaymentID) AS TOTAL_COUNT ,paymentmethod.PaymentMethodCode from payment
inner join paymentmethod on payment.PaymentMethodID = paymentmethod.PaymentMethodID
group by paymentmethod.PaymentMethodID;
```

- Gives the total count of the different payment methods to the airline management.