**Design a Database for Flight Booking System from Scratch**

**Background:**

The procedure of making a flight reservation is a complicated one in today’s society. Airlines require a method of managing everything in order to handle bookings from eager passengers on a continuous basis. The way records are kept has changed significantly over time as a result of technical advancements, such as the introduction of computers, which allowed airlines to abandon manual procedures.

**Objectives:**

a. Keeping track of and securely keeping booking records.  
b. The database should make it simple to recover and retrieve booking information.  
c. Data loss will be prevented, and data integrity will be upheld, removing the possibility of data corruption.  
d. Reducing data entry errors that could have an impact on the data that’s stored in the system.  
e. Only authorized users will be able to access the information via the automated flight booking system.  
f. Reduce duplication by keeping a centralized database and only saving data once.  
g.Reduces paperwork.

**Purpose:**

This project’s main objective is to save paperwork while still giving consumers a straightforward way to look for and book flights. Users can check their tickets and reservations, receive information on ticket prices, and retrieve data on several flights that are available at various times on a given date using this database. It stores all the information needed to plan a trip, including the source, destination, and any layovers.  
It will store, organize, and manage a sizable amount of data, including a database of multiple flights, in order to maintain a vast scale of data.

**Initial ERD Design:**

**Discussed Business Problem :**

1. A Flight Booking System will keep a track and maintain booking of records efficiently and safely.
2. Booking details should be easily recoverable and retrievable from the database.
3. Airlines need to constantly process bookings from expectant passengers, they need a means of keeping on top of everything. Therefore, storing the records of flight schedules and passenger details will keep the data organized.
4. The automated flight booking system will allow only authorized users to access the information.
5. The system reduces redundancy by maintaining a centralized database and storing all information only once.

**Entities Present:**

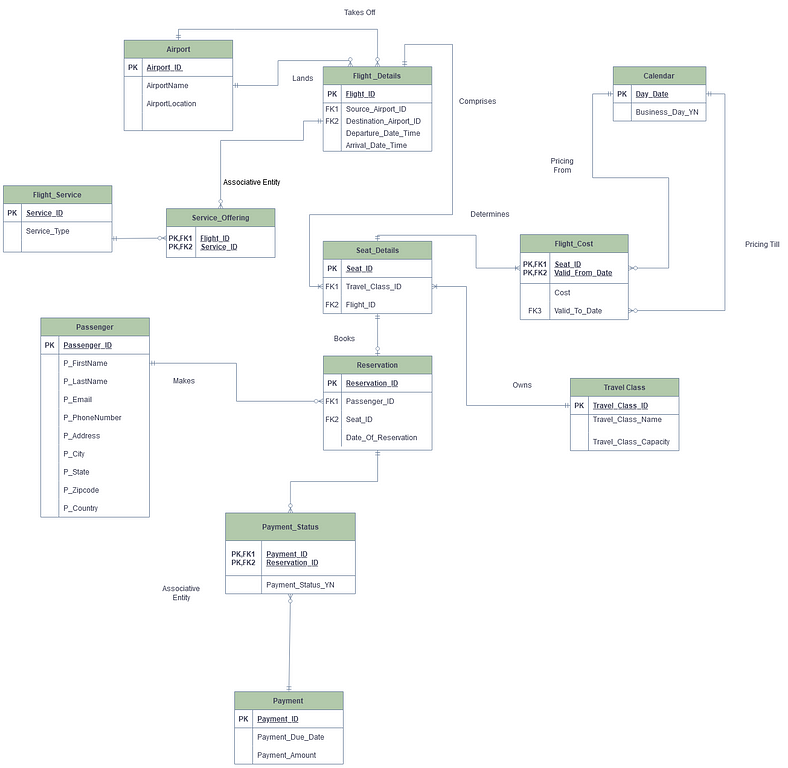
1. Airport
2. Passenger
3. Flight\_Details
4. Flight\_Service
5. Seat\_Details
6. Flight\_Cost
7. Reservation
8. Travel\_Class
9. Calendar
10. Payment
11. Payment\_Status
12. Service\_Offering

**Relationships between entities:**

1. An airport manages multiple flights. An airport may or may not have a flight landing/takeoff. Hence, it’s optional. Whereas for flight details, it is mandatory to have one airport association.
2. A flight may offer multiple services and each service may be offered to many flights.
3. A flight comprises multiple seats. It is mandatory that the flight consists of a seat. For Seat details to exist, it is mandatory that it is linked to at least one flight.
4. A passenger can make multiple reservations for a flight. Passengers may or may not reserve a flight. However, it is mandatory that reservation has to be associated with only one passenger.
5. Reservation confirms the Seat Details. It is mandatory that a reservation is linked with a Seat. A seat may or may not be linked with a reservation. One seat can only be linked to one reservation.
6. A travel class should consist of multiple seats. A seat should be linked to only one travel class.
7. A Payment may consist of multiple payment status. Whereas, a payment status should have a Payment associated with it.
8. A Payment Status should have a Reservation. A reservation may consist of multiple payment status.
9. The Flight cost should be determined from the Calendar.
10. The Seat Details should be associated with a Flight cost. The Flight Cost is calculated based on the Seat Details.

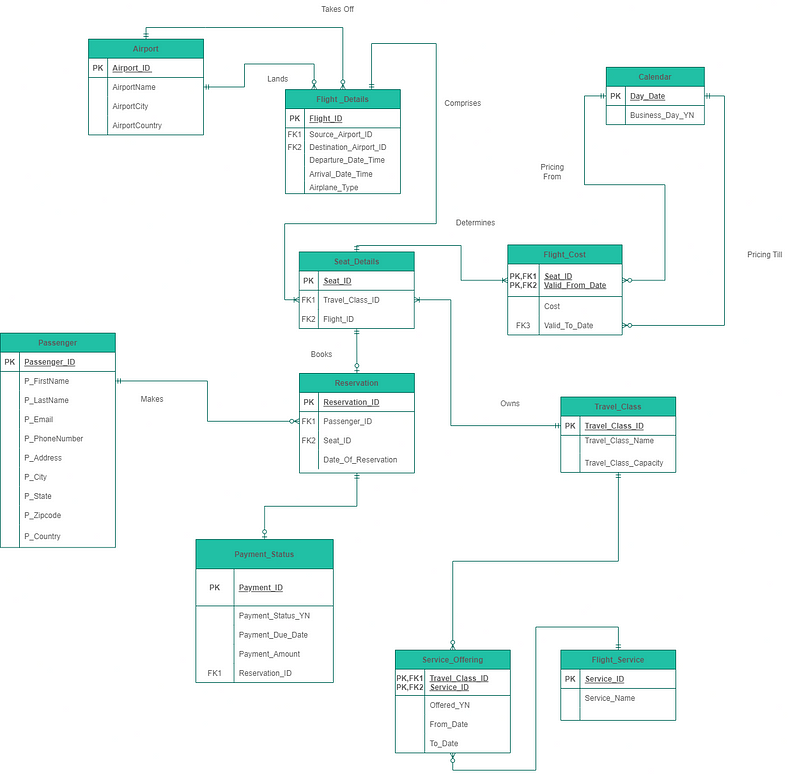
**Key Design:**

1. Each Airport can have multiple flight details associated with its respective source and destination.
2. Each Flight may offer many flight services like Food Service, Entertainment Service, Wifi Service etc. Each Flight\_Service may be offered to many flights. Service\_Offering is an associative entity which stores Service\_ID and Flight\_ID.
3. Each Flight mandatorily comprises one or many seats and each seat is associated with a particular Travel\_Class\_ID. A seat can belong to Business class or economy class etc.
4. Each passenger may or may not make multiple reservations. If a passenger makes a reservation, a seat of the passenger’s choice will be assigned. The Seat\_Details will point to Travel\_Class and Flight\_Details. For each seat reserved, a passenger may or may not pay to confirm the seat. A passenger can reserve one seat per Reservation\_ID.
5. Payment\_Status is an associative entity which holds the information related to particular Reservation\_ID and Payment\_ID.
6. If the passenger has or hasn’t done the payment it will reflect in payment status and the respective payment due date and amount will be stored in Payment.
7. Flight\_Cost can be determined on the basis of Seat\_Details and Calendar from which a particular seat belongs to a travel class and the date when the seat is booked respectively.



**Final ERD Design:**

This intial ER Diagram can be improved by removing redundancies and refining the entities



· Airport entity has AirportCity and AirportCountry which provides the airport specifications.

· Flight\_Type attribute has been added to Flight\_Details

· Payment entity has been removed from the ER diagram and its attributes have been listed under Payment\_Status.

· The cardinality between Payment\_Status and Reservation has been made optional one to mandatory many. Therefore, Reservation\_ID becomes foreign key inside Payment\_Status.

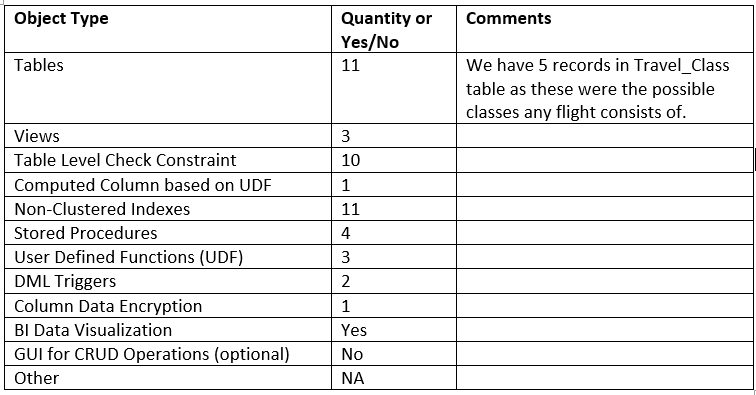
· Service\_Offering which is an associative entity has 3 new attributes called Offered\_YN , From\_Date and To\_Date. From\_Date and To\_Date that would give us the period when certain flight services are provided. Offered\_YN would tell us if that service is been offered to a particular travel class.

· From the previous ER Diagram, we have removed Service\_Type from Flight\_Service and Class\_Type from Travel\_Class as it’s not required to enhance it.

**Database Implementation:**

With the Data Model ready,let us implement it using SQL.I have chosen Microsoft SQL server for the implementation

Here is a configured list of every object used in this project



let’s look at the DDL script first.

**DDL:**

go  
Create Database FBS;  
go  
USE FBS  
--Create Airport table  
CREATE TABLE Airport  
(  
Airport\_ID int not null,  
AirportName varchar(100),  
AirportCity varchar(100),  
AirportCountry varchar(100),  
CONSTRAINT Airport\_PK PRIMARY KEY (Airport\_ID)  
);  
  
  
-- create Passenger table   
-- new changes  
create table Passenger   
(  
Passenger\_ID INT not null,  
P\_FirstName VARCHAR(100),  
P\_LastName VARCHAR(100),  
P\_Email VARCHAR(100) CONSTRAINT email\_check CHECK (P\_Email LIKE '[a-z,0-9,\_,-]%@[a-z]%.[a-z][a-z]%'),  
P\_PhoneNumber BIGINT not null UNIQUE CONSTRAINT Ph\_length\_check CHECK (len([P\_PhoneNumber])=10),  
P\_Address VARCHAR(100),  
P\_City VARCHAR(100),  
P\_State VARCHAR(100),  
P\_Zipcode VARCHAR(100) CONSTRAINT zip\_chk CHECK (LEN(P\_Zipcode)=5),  
P\_Country VARCHAR(100)  
CONSTRAINT Passenger\_ID\_PK PRIMARY KEY (Passenger\_ID)  
);  
  
-- create Travel class  
CREATE TABLE Travel\_Class  
(  
 Travel\_Class\_ID INT NOT NULL,  
 Travel\_Class\_Name VARCHAR(100) CONSTRAINT name\_list\_chk CHECK (Travel\_Class\_Name IN('First Class','Business Class','Premium Economy','Economy Class','Basic Economy')),  
 Travel\_Class\_Capacity BIGINT,  
 CONSTRAINT Travel\_Class\_PK PRIMARY KEY (Travel\_Class\_ID)  
);  
  
-- create Calendar table  
CREATE TABLE Calendar  
(  
 Day\_Date Date NOT NULL,  
 Business\_Day\_YN CHAR(1) Constraint check\_character\_Business\_Day\_YN Check(Business\_Day\_YN In ('Y','N')),  
 CONSTRAINT Calendar\_PK PRIMARY KEY (Day\_Date)  
);  
  
-- create Flight Service table  
create table Flight\_Service  
(  
Service\_ID INT not null,  
[Service\_Name] VARCHAR(100) CONSTRAINT Service\_chk CHECK([Service\_Name] in ('Food','French Wine','Wifi','Entertainment','Lounge')),  
CONSTRAINT Flight\_Service\_PK PRIMARY KEY (Service\_ID)  
);  
  
--ALTER TABLE Flight\_Service ALTER [Service\_Name] CONSTRAINT Service\_chk CHECK([Service\_Name] in ('Food','French Wine','Wifi','Entertainment','Lounge'));  
-- create Flight Details table  
CREATE TABLE Flight\_Details (  
 Flight\_ID INT NOT NULL,  
 Source\_Airport\_ID INT NOT NULL,  
 Destination\_Airport\_ID INT NOT NULL ,  
 Departure\_Date\_Time DateTime,-- CONSTRAINT date\_check CHECK (Departure\_Date\_Time< Arrival\_Date\_Time),  
 Arrival\_Date\_time DateTime,  
 Airplane\_Type VARCHAR(100) CONSTRAINT airplane\_check CHECK(Airplane\_Type IN ('Airbus A380','Boeing 747')),  
 CONSTRAINT Flight\_Details\_PK PRIMARY KEY (Flight\_ID),  
 CONSTRAINT Flight\_Details\_Source\_FK1 FOREIGN KEY (Source\_Airport\_ID) REFERENCES Airport(Airport\_ID),  
 CONSTRAINT Flight\_Details\_Destination\_FK2 FOREIGN KEY (Destination\_Airport\_ID) REFERENCES Airport(Airport\_ID),  
 CONSTRAINT Date\_check\_FD CHECK (Departure\_Date\_Time< Arrival\_Date\_Time),  
 CONSTRAINT airport\_chk CHECK (Source\_Airport\_ID != Destination\_Airport\_ID)  
 );  
  
 -- create Seat Details table  
 CREATE TABLE Seat\_Details  
(  
 Seat\_ID VARCHAR(100) NOT NULL,  
 Travel\_Class\_ID INT NOT NULL,   
 Flight\_ID INT NOT NULL,  
 CONSTRAINT Seat\_Details\_PK PRIMARY KEY (Seat\_ID),  
 CONSTRAINT Seat\_Details\_TravelClassID\_FK1 FOREIGN KEY (Travel\_Class\_ID) REFERENCES Travel\_Class(Travel\_Class\_ID),  
 CONSTRAINT Seat\_Details\_FlightID\_FK2 FOREIGN KEY (Flight\_ID) REFERENCES Flight\_Details(Flight\_ID)  
)  
  
-- create Reservation table   
 CREATE TABLE Reservation (  
 Reservation\_ID INT NOT NULL,  
 Passenger\_ID INT NOT NULL,  
 Seat\_ID VARCHAR(100) NOT NULL,  
 --Default Value recorded below  
 Date\_Of\_Reservation Date DEFAULT(getDate()),  
 CONSTRAINT Reservation\_PK PRIMARY KEY (Reservation\_ID),  
 CONSTRAINT Reservation\_Passenger\_ID\_FK1 FOREIGN KEY (Passenger\_ID) REFERENCES Passenger(Passenger\_ID),  
 CONSTRAINT Reservation\_Seat\_ID\_FK2 FOREIGN KEY (Seat\_ID) REFERENCES Seat\_Details(Seat\_ID)  
 );  
  
 -- create Payment Status table  
 --add table again  
 CREATE TABLE Payment\_Status (  
 Payment\_ID INT NOT NULL IDENTITY(1,1),  
 Payment\_Status\_YN CHAR(1) Constraint check\_character\_Payment\_Status\_YN Check(Payment\_Status\_YN In ('Y','N')),  
 Payment\_Due\_Date Date,  
 Payment\_Amount INT,  
 Reservation\_ID INT NOT NULL,  
 CONSTRAINT Payment\_Status\_PK PRIMARY KEY (Payment\_ID),  
 CONSTRAINT Payment\_Reservation\_ID\_FK FOREIGN KEY (Reservation\_ID) REFERENCES Reservation(Reservation\_ID)  
 );  
  
--create Flight cost table   
CREATE TABLE Flight\_Cost  
(   
 Seat\_ID VARCHAR(100) NOT NULL,  
 Valid\_From\_Date Date NOT NULL,  
 Valid\_To\_Date Date NOT NULL,  
 Cost BIGINT,  
 CONSTRAINT Flight\_Cost\_PK PRIMARY KEY (Seat\_ID,Valid\_From\_Date),  
 CONSTRAINT Flight\_Cost\_Seat\_ID\_FK1 FOREIGN KEY (Seat\_ID) REFERENCES Seat\_Details(Seat\_ID),  
 CONSTRAINT Flight\_Cost\_Valid\_From\_Date\_FK2 FOREIGN KEY (Valid\_From\_Date) REFERENCES Calendar(Day\_Date),  
 CONSTRAINT Flight\_Cost\_Valid\_To\_Date\_FK3 FOREIGN KEY (Valid\_To\_Date) REFERENCES Calendar(Day\_Date)  
);  
  
--create Service offering table   
CREATE TABLE Service\_Offering  
(  
 Travel\_Class\_ID INT NOT NULL,  
 Service\_ID int NOT NULL,  
 Offered\_YN CHAR(1) Constraint check\_character\_Offered\_YN Check(Offered\_YN In ('Y','N')),  
 From\_Month VARCHAR(20),  
 To\_Month VARCHAR(20),  
 CONSTRAINT Service\_Offering\_TCI\_FK1 FOREIGN KEY (Travel\_Class\_ID) REFERENCES Travel\_Class(Travel\_Class\_ID),  
 CONSTRAINT Service\_Offering\_SID\_FK2 FOREIGN KEY (Service\_ID) REFERENCES Flight\_Service(Service\_ID),  
 CONSTRAINT Service\_Offering\_PK PRIMARY KEY (Travel\_Class\_ID,Service\_ID)  
);  
  
--Creating indexes for all primary keys  
CREATE UNIQUE INDEX SO\_Index ON Service\_Offering(Travel\_Class\_ID,Service\_ID);  
CREATE UNIQUE INDEX FC\_Index ON Flight\_Cost(Seat\_ID,Valid\_From\_Date);  
CREATE UNIQUE INDEX PS\_Index ON Payment\_Status(Payment\_ID);  
CREATE UNIQUE INDEX R\_Index ON Reservation(Reservation\_ID);  
CREATE UNIQUE INDEX SD\_Index ON Seat\_Details(Seat\_ID);  
CREATE UNIQUE INDEX FD\_Index ON Flight\_Details(Flight\_ID);  
CREATE UNIQUE INDEX FS\_Index ON Flight\_Service(Service\_ID);  
CREATE UNIQUE INDEX Cal\_Index ON Calendar(Day\_Date);  
CREATE UNIQUE INDEX T\_Index ON Travel\_Class(Travel\_Class\_ID);  
CREATE UNIQUE INDEX Pass\_Index ON Passenger(Passenger\_ID);  
CREATE UNIQUE INDEX Air\_Index ON Airport(Airport\_ID);  
  
--Creating non clustered index  
USE P4\_FBS  
CREATE NONCLUSTERED INDEX email\_indx ON Passenger(P\_Email);  
CREATE NONCLUSTERED INDEX dep\_indx ON Flight\_Details(Departure\_Date\_Time);  
CREATE NONCLUSTERED INDEX city\_indx ON Airport(AirportCity);  
go  
sp\_helpindex Airport  
go  
sp\_helpindex Passenger  
go  
sp\_helpindex Flight\_Details

Let us setup triggers and necessary functions to auto insert data and calculate flight costs based on the dates

go  
USE FBS  
go  
CREATE TRIGGER Payment\_Insert ON Reservation  
--MAKE IDENTITY FOR PAYMENT ID  
  
FOR INSERT  
AS  
BEGIN  
 DECLARE @ResId int  
 DECLARE @Res\_date DATE  
 DECLARE @due\_date DATE  
 DECLARE @Cost bigint  
 DEClARE @seat VARCHAR(20)   
 SELECT @ResId = Reservation\_ID FROM inserted  
 SELECT @Res\_date = Date\_Of\_Reservation FROM inserted  
 SELECT @seat= Reservation.Seat\_ID FROM Reservation WHERE Reservation\_ID=@ResId  
 SELECT @Cost=Cost FROM Flight\_Cost WHERE Flight\_Cost.Seat\_ID=@seat AND Flight\_Cost.Valid\_From\_Date=@Res\_date  
 SELECT @due\_date=DATEADD(DAY,1,@Res\_date)  
 INSERT INTO Payment\_Status(Payment\_Status\_YN,Payment\_Due\_Date,Payment\_Amount,Reservation\_ID)  
 VALUES ('N',@due\_date,@Cost,@ResId);  
  
  
END  
  
  
  
--UDF  
--OUR UDF WILL TAKE BUSINESS DAY,Departure Date,Travel cLASS id,Deafult price  
go  
CREATE FUNCTION Cal\_Cost(@Start\_Date date,@dep\_date date,@travel\_id int,@business char(1))  
RETURNS BIGINT  
AS  
BEGIN  
 DECLARE @Diff int  
 Select @Diff=DateDIFF(day,@Start\_Date,@dep\_date)  
 RETURN 100+((Select  
 CASE  
 WHEN @business='Y' THEN 10  
 ELSE 0  
 END As Business\_Inc)+  
 (Select  
 CASE   
 WHEN @travel\_id=1 THEN 50  
 WHEN @travel\_id=2 THEN 40  
 WHEN @travel\_id=3 THEN 30  
 WHEN @travel\_id=4 THEN 20  
 ELSE 10  
 END As Class\_Inc)+  
 (Select  
 CASE   
 WHEN @Diff>20 AND @Diff<=30 THEN 0  
 WHEN @Diff>10 AND @Diff<=20 THEN 20  
 ELSE 50  
 END AS Interval\_Inc  
 ))  
   
   
END  
  
  
--function body end  
  
GO  
CREATE TRIGGER Set\_Cost ON Seat\_Details  
FOR INSERT  
AS  
BEGIN  
 Declare @Default\_price BIGINT  
 Set @Default\_price=100  
 DECLARE @Start\_date date  
 DECLARE @Final\_Cost BIGINT  
 Declare @business char(1)  
 --Declare @default\_price bigint  
 Declare @flight\_cost bigint  
 DECLARE @dep\_date date  
 --set @default\_price=100  
 Declare @S\_id varchar(20)  
 SELECT @S\_id = Seat\_ID FROM inserted  
 Declare @travel\_id int  
 SELECT @travel\_id= Travel\_Class\_ID FROM inserted  
 DECLARE @flight\_id int  
 SELECT @flight\_id=Flight\_ID FROM inserted  
   
 Select @business=Business\_Day\_YN FROM Calendar  
 WHERE Day\_Date=  
 (SELECT CONVERT(Date,Departure\_Date\_Time)   
 FROM Flight\_Details WHERE Flight\_ID=@flight\_id  
 )-- from Flight\_Details where Flight\_ID=@flight\_id  
 SELECT @dep\_date=CONVERT(Date,Departure\_Date\_Time)   
 FROM Flight\_Details WHERE Flight\_ID=@flight\_id  
  
 SELECT @Start\_date=DATEADD(MONTH,-1,@dep\_date)  
 DECLARE @last\_date date  
 SELECT @last\_date=DateADD(day,-1,@dep\_date)  
-- WHILE LOOP  
   
 WHILE @Start\_date < @last\_date  
 BEGIN  
  
 SELECT @Final\_Cost=[dbo].[Cal\_Cost](@Start\_date,@dep\_date,@travel\_id,@business)  
 DECLARE @valid\_to date  
 SET @valid\_to= DATEADD(Day,1,@Start\_date)  
 INSERT INTO Flight\_Cost(Seat\_ID,Valid\_From\_Date,Cost,Valid\_To\_Date)  
 VALUES(@S\_id,@Start\_date,@Final\_Cost,@valid\_to)  
 SET @Start\_date = DATEADD(Day,1,@Start\_date)  
 END  
  
END

Implementing Stored Procedures to improve functionality of our database and make it easier for developers to access the database and make any changes

USE FBS  
--procedure 1  
Go  
CREATE PROCEDURE Flight\_Availability  
@Departure\_Date date,  
@Arrival\_Date date,  
@available char(1) OUTPUT  
  
AS  
IF EXISTS(SELECT Flight\_ID FROM Flight\_Details  
 WHERE CONVERT(Date,Departure\_Date\_time) = @Departure\_Date  
 AND CONVERT(Date,Arrival\_Date\_time) = @Arrival\_Date   
)  
  
BEGIN  
 SET @available='Y'   
 SELECT \* FROM Flight\_Details  
 WHERE CONVERT(Date,Departure\_Date\_time) = @Departure\_Date  
 AND CONVERT(Date,Arrival\_Date\_time) = @Arrival\_Date  
   
END  
ELSE  
BEGIN  
  
Print 'Flight Detail Doesnt exist'  
end  
  
  
  
  
  
  
  
  
--PROCEDURE 2  
  
Go  
CREATE PROCEDURE [dbo].[UpdateFlightDetails]   
@flag bit output,-- return 0 for fail,1 for success  
@FlightID INT,  
@DeptDateTime DATETIME,  
@ArivalDateTime DATETIME,  
@AirplaneType VARCHAR(100)  
AS  
BEGIN  
 Update Flight\_Details set Departure\_Date\_Time = @DeptDateTime,  
 Arrival\_Date\_time = @ArivalDateTime,  
 Airplane\_Type = @AirplaneType  
 Where Flight\_ID = @FlightID   
 set @flag=1;  
   
END  
  
  
  
 --Procedure 3  
  
 Go  
 CREATE PROCEDURE [dbo].[PassengerCRUD]   
 @Action VARCHAR(10),   
 @Passenger\_ID int = NULL,   
 @FName VARCHAR(100) = NULL,   
 @LName VARCHAR(100) = NULL,   
 @Email VARCHAR(100) = NULL,  
 @PNumber BIGINT = NULL,   
 @Address VARCHAR(100) = NULL,  
 @city VARCHAR(100) = NULL,   
 @State VARCHAR(100) = NULL,   
 @Zipcode VARCHAR(100) = NULL,   
 @Country VARCHAR(100) = NULL  
  
AS  
BEGIN  
 SET NOCOUNT ON;  
   
 --SELECT  
 IF @Action = 'SELECT'  
 BEGIN  
 SELECT Passenger\_ID, P\_FirstName,P\_LastName,P\_Email,P\_PhoneNumber,P\_Address,P\_City,  
 P\_State,P\_Zipcode,P\_Country  
 FROM Passenger  
 END  
   
 --INSERT  
 IF @Action = 'INSERT'  
 BEGIN  
 INSERT INTO dbo.Passenger([Passenger\_ID], [P\_FirstName],[P\_LastName],[P\_Email],[P\_PhoneNumber],[P\_Address],[P\_City],  
 [P\_State],[P\_Zipcode],[P\_Country] )  
  
 VALUES (@Passenger\_ID, @FName, @LName, @Email, @PNumber, @Address, @city, @State, @Zipcode, @Country)  
 END  
   
 --UPDATE  
 IF @Action = 'UPDATE'  
 BEGIN  
 UPDATE Passenger  
 SET Passenger\_ID = @Passenger\_ID, P\_FirstName = @FName, P\_LastName = @LName, P\_Email= @Email, P\_PhoneNumber = @PNumber,   
 P\_Address = @Address, P\_City = @city, P\_State = @State, P\_Zipcode = @Zipcode, P\_Country = @Country  
 WHERE Passenger\_ID = @Passenger\_ID  
 END  
   
 --DELETE  
 IF @Action = 'DELETE'  
 BEGIN  
 DELETE FROM Passenger  
 WHERE Passenger\_ID = @Passenger\_ID  
 END  
END  
  
  
  
--Procedure 4  
GO  
CREATE PROCEDURE UpdatePayment   
 @paymentID INT,@paid\_date date OUTPUT  
AS  
 BEGIN  
 Update Payment\_Status  
 SET Payment\_Status\_YN='Y' WHERE Payment\_ID=@paymentID  
 SELECT @paid\_date=GETDATE()  
   
END

Finally insert valid data using DML to manipulate and get insights

**Execution:**

The right way to set this up is by running DDL,Trigger Script,Stored Procedures and finally your DML script.

**Visualization:**

Finally, with my inserted data I have used Tableau to visualize and infer certain insights

