DOMESTIC AIRLINE RESERVATION SYSTEM

Team 6

Dedeepya Chintaparthi - 801056005 Keyur Hansoti – 801044954
Rachana Revuri – 801082632 Renju Hanna Robin - 801076715

PROJECT PROPOSAL

Context:

The project is proposed in the context of Domestic Airline Reservation System. In this system, User can search for the availability of flights, Check flight schedules and Fare. The users can book a flight, make a payment, cancel a ticket and reschedule the flight. The user will add his/her personal information while making a payment.

Scope and Objectives:

The project is to develop a database system that contains the data of proposed Domestic Airline Reservation System. A user interface will be developed as part of the project so that the user can interact with the application through the UI.

In this system, there are two types of users namely Guest and Registered user. Registered user will be able to check the availability of flights. A registered user can book a flight, make a payment, cancel a ticket and reschedule the flight. A guest user can search for the availability of flight, check flight schedules and fare.

Apart from the users there will be an Admin who can manage the application and the database system.

PROJECT ENVIRONMENT

The application will be a web based application built using PHP and HTML with MySQL. For the development we are using WampServer 2.2 which has

- MySQL 5.5.24
- PHP 5.3.13
- Apache server 2.2.22
- PHPMyAdmin 3.5.1
- XDebug 2.2.0

For coding we are using the NetBeans IDE 8.1

HIGH LEVEL REQUIREMENTS

Initial user roles

User Role	Description
Guest User	A guest user can search for the availability of flight, check flight schedules and fare.
Registered user	A registered user can check the availability of flights, book a flight, make a payment, cancel a ticket and reschedule the flight.
Administrator	Admin who can manage the application and the database system.

High level user story descriptions:

Initial user story descriptions

Story ID	Story description
US1	As an Administrator, I want to log in to the system.
US2	As an Administrator, I want to add flight details.
US3	As a Guest User, I want to search for availability of flights.
US4	As a Guest User, I want to check flight schedule.
US5	As a Guest User, I want to check flight fare.
US6	As an Guest user, I want to register so that I can book flight.
US7	As a Registered User, I want to log in to the system.
US8	As a Registered User, I want to search for availability of flights.
US9	As a Registered User, I want to check flight schedule.
US10	As a Registered User, I want to check flight fare.
US11	As a Registered User, I want to book flight.

US12	As a Registered User, I want to reschedule my flight.
US13	As a Registered User, I want to cancel my flight.

HIGH LEVEL CONCEPTUAL DESIGN

Entities:

Guest_User
Registered_User
Flight
Administrator

Relationships:

Administrator *adds* **Flight** details.
Guest_User *searches* for **Flight**.
Registered_User *searches* for **Flight.**
Registered_User *books* **Flight.**
Registered_User *reschedule* **Flight.**
Registered_User *cancels* ticket of the **Flight.**

Sprint 1

REQUIREMENTS

General Notes:

- We assume that Guest Users don't need an account as they are just users who can access some basic features without signing up. So we are not storing the data related to Guest User.
- At this time, data about searches conducted by users need not be recorded.

Story ID	Story description
US1	As an Administrator, I want to add flight details.
US2	As a User, I want to search for availability of flights.
US3	As a User, I want to check flight schedule.
US4	As a User, I want to check flight fare.
US5	As an Guest user, I want to register so that I can book flight.
US6	As a Registered User, I want to book flight.
US7	As a Registered User, I want to reschedule my flight.
US8	As a Registered User, I want to cancel my flight.
US9	As an Administrator , I want to update the flight details.
US10	As a Registered User, I want to book roundtrip as well as one-way flight.
US11	As a Registered User, I want to book flight for multiple passengers.
US12	As a User can, I want to give a feedback.
US13	As an Administrator, I want to see user feedback.

The role name "User" implies both Guest and Registered Users.

Refined Requirements:

1. As an Administrator, I want to add the flight details so that user can view and search for them.
2. As a User, I want to perform a search for flight availability between two cities for a given date so that I can check flight schedule and fare
   Notes : We have combined the user stories US2, US3 and US4

3. As a User I want to register so that i can become a Registered User and perform flight booking.
4. As a Registered User , I want to book flight of particular Airline of my choice.
   Notes :
   - Book is generic in practical scenario, hence we incorporate the airline preferences of the users.

CONCEPTUAL DESIGN

Entity: **Flight**

Attributes:

flight_id[Atomic]

flight_number [Atomic]

flight_status [Atomic]

flight_type [Atomic]

flight_origin [Atomic]

flight_destination [Atomic]

arrival_time [Atomic]

departure_time [Atomic]

flight_date[Atomic]

flight_fare[Atomic]

Entity: **Airline**

Attributes:

airline_code [Atomic]

airline_name [Atomic]

Entity: **Registered_User**

Attributes:

- user_name [Atomic]
- password[Atomic]
- name [composite]
  - first_name
  - last_name
- email[Atomic]
- address [composite]
  - address_line_1
  - address_line_2
  - city
  - state
  - zip_code
- phone_number[Multi_valued]
- credit_card_information[composite]
  - cc_name
  - cc_num
  - cc_expiry
  - cc_cvv

Entity: **Administrator**

Attributes:

- admin_id [Atomic]
- username[Atomic]
- password[Atomic]

Action: As an Guest user, I want to register so that I can book a flight.

Relationship: **Registered_User** books **Flight** of particular **Airline**

Cardinality : Many to Many (**Registered_User** books **Flight)**
Many to One (**Flight** of particular **Airline)**

Participation:

Registered_User has partial participation

Flight has total participation (In both the relationships)

Airline has total participation

Relationship: **Administrator** adds **Flight**

Cardinality : One to Many

Participation:

Administrator has partial participation

Flight has total participation

LOGICAL DESIGN

Table: **Flight**

Columns:

flight_id (Primary Key)

flight_number

flight_status

flight_type

flight_origin

flight_destination

arrival_time

departure_time

flight_date

flight_fare

airline_code(Foreign key:references airline_code of Airline)

admin_id (Foreign key:references admin_id of Administrator)

Justification : admin_id added as an foreign key to represent the relationship Administrator adds flight (One to Many).

Also in an airlines systems , we can have more than one admin as it is a very large database .So we assume that we have more than one admin in our case.

Table: **Airline**

Columns:

airline_code (Primary Key)

airline_name

Justification:airline_code is unique for every airline, hence it is chosen as the primary key.

Table: **Registered_User**

Columns:

user_name (Primary Key)

password

first_name

last_name

email

address_line_1

address_line_2

zip_code (Foreign key:references zip_code of Zip_Codes table)

phone_number_primary

phone_number_secondary

Justification:user_id is unique for every user, hence it is chosen as the primary key for Registered_User.

Table: **Card_Details**

Columns:

cc_num (Primary key)

cc_name

cc_expiry

cc_cvv

user_name (Foreign key:references user_name of Registered_User)

Justification: There is a possibility that we can have duplicate data for card details for an user ,i.e.when tickets are booked multiple times by the user.To overcome this we separate the details of the card from the Registered_User by creating a separate table called Card_Details and we add the user_id from Registered_User table as a foreign key to the Card_Details table.

Table: **Zip_Codes**

Columns:

zip_code (Primary Key)

city

state

Justification: There is a possibility that we can have duplicate data for zip code details of users, i.e. when different users of same city or state register. To overcome this we separate the details of the zip_codes from the Registered_User by creating a separate table called Zip_Codes and we add the zip_code primary key from Zip_Codes as a foreign key to the Registered_User table .

Table: **Administrator**

Columns:

admin_id (Primary Key)

username

password

Justification: admin_id is unique for every admin user, hence it is chosen as the primary key for Administrator.

Table: **UserBookFlight**

Columns:

booking_id (Primary key)

user_name (Foreign key:references user_name of Registered_User)

flight_id (Foreign key : references flight_id of Flight)

Primary key justification: In this case, booking_id suffices as a primary key .

Justification : We have a ternary relationship called "**Registered_User** books **Flight** of particular **Airline"** in our conceptual design.To map this to the logical design we are creating a seperate table called **UserBookFlight** by using cross-reference approach.

Sprint 2

REQUIREMENTS

General Notes:

- We assume that Guest Users don't need an account as they are just users who can access some basic features without signing up. So we are not storing the data related to Guest User.
- At this time, data about searches conducted by users need not be recorded.

Story ID	Story description
US1	As an Administrator, I want to add flight details.
US2	As a User, I want to search for availability of flights.
US3	As a User, I want to check flight schedule.
US4	As a User, I want to check flight fare.
US5	As an Guest user, I want to register so that I can book flight.
US6	As a Registered User, I want to book flight.
US7	As a Registered User, I want to reschedule my flight.
US8	As a Registered User, I want to cancel my flight.

US9	As an Administrator , I want to update the flight details.
US10	As a Registered User, I want to book a flight for round trip as well.
US11	As a Registered User, I want to book flight for multiple passengers.
US12	As a User can, I want to give a feedback.
US13	As an Administrator, I want to see user feedback.

The role name "User" implies both Guest and Registered Users.

Refined Requirements:

1. As an Administrator, I want to add the flight details so that user can view and search for them.
2. As a User, I want to perform a search for flight availability between two cities for a given date so that I can check flight schedule and fare

   Notes : We have combined the user stories US2, US3 and US4

3. As a User I want to register so that i can become a Registered User and perform flight booking.
4. As a Registered User , I want to book flight of particular Airline of my choice.

   Notes :

   - Book is generic in practical scenario, hence we incorporate the airline preferences of the users.

5. As an Administrator, I want to update the flight details so that I can update the flight which got delayed or canceled.
6. As an Registered user, I want to reschedule or cancel my flight.

   Notes :

   In general, rescheduling a flight implies cancelling one flight and booking another. So, we have combined the user stories US7 and US8

NOTES : we are not storing any details regarding the user cancelling or rescheduling the flights and administrator updating the flight details in the database.

CONCEPTUAL DESIGN

Entity: **Flight**

Attributes:

flight_id[Atomic]

flight_number [Atomic]

flight_status [Atomic]

class_type[Atomic]

flight_origin [Atomic]

flight_destination [Atomic]

arrival_time [Atomic]

departure_time [Atomic]

flight_date[Atomic]

flight_fare[Atomic]

flight_delay[Atomic]

no_of_seats [Atomic]

Entity: **Airline**

Attributes:

airline_code [Atomic]

airline_name [Atomic]

Entity: **Registered_User**

Attributes:

user_name [Atomic]

password[Atomic]

name [composite]

first_name

last_name

email[Atomic]

address [composite]

address_line_1

address_line_2

city

state

zip_code

phone_number[Multi_valued]

credit_card_information[composite]

cc_name
cc_num
cc_expiry
cc_cvv

Entity: **Administrator**

Attributes:

admin_id [Atomic]

username[Atomic]

password[Atomic]

We are assuming Administrator as virtual, so no other details like his name or address are saved in Database.

Action: As an Guest user, I want to register so that I can book a flight.

Relationship: **Registered_User** can book **Flight.**

Cardinality : Many to Many

Participation:

Registered_User has partial participation

Flight has total participation

Relationship: **Flight** belongs to **Airline**

Cardinality : Many to One

Participation:

Flight has total participation

Airline has total participation

Relationship: **Administrator** adds **Flight.**

Cardinality : One to Many

Participation:

Administrator has total participation

Flight has total participation

LOGICAL DESIGN WITH NORMALISATION

Table: **Flight**

Columns:

flight_id (Primary Key)

flight_number

flight_date

airline_code(Foreign key:references airline_code of Airline)

admin_id (Foreign key:references admin_id of Administrator)

Justification : admin_id added as an foreign key to represent the relationship Administrator adds flight (One to Many).

airline_code is added as an foreign key to represent the relationship Flight belongs to Airline (Many to One).

Also in an airlines systems, we can have more than one admin as it is a very large database .So we assume that we have more than one admin in our case.

Highest normalization level: 4NF

Table: **Flight_Fare**

Columns:

flight_id (Foreign key: references flight_id of Flight table)

flight_origin

flight_destination

flight_fare

Highest normalization level: 4NF

Table: **Flight_Status**

Columns: flight_id (Foreign key: references flight_id of Flight table)
flight_status
flight_delay
arrival_time
departure_time

Justification : flight_delay column is added to show the delayed time of a flight with flight_status='Delayed'.

Highest normalization level: 4NF

Table: **Flight_Class**

Columns:

flight_id (Foreign key: references flight_id of Flight table)

class_type

no_of_seats

Justification : class_type column distinguishes between the economy class and business class of the flight.

Highest normalization level: 4NF

Table: **Airline**

Columns:

airline_code (Primary Key)

airline_name

Justification:airline_code is unique for every airline, hence it is chosen as the primary key.

Highest normalization level: 4NF

Table: **Registered_User**

Columns:

user_name (Primary Key)

password

first_name

last_name

email

address_line_1

address_line_2

zip_code (Foreign key:references zip_code of Zip_Codes table)

phone_number_primary

phone_number_secondary

Justification: user_name is unique for every user, hence it is chosen as the primary key for Registered_User.

Highest normalization level: 4NF

Table: **Card_Details**

Columns:

cc_num (Primary Key)

cc_name

cc_expiry

cc_cvv

user_name (Foreign key:references user_name of Registered_User)

Justification:

1. There is a possibility that we can have duplicate data for card details for an user ,i.e.when tickets are booked multiple times by the user.To overcome this we separate the details of the card from the Registered_User by creating a separate table called Card_Details and we add the user_id from Registered_User table as a foreign key to the Card_Details table.

2. A user can use multiple cards for different bookings, hence we username cannot determine any other column. So the table is already in BCNF.

No multivalued dependency between keys, hence the table is finally in 4NF

Highest normalization level: 4NF

Table: **Zip_Codes**

Columns:

zip_code (Primary Key)

city

state

Justification: There is a possibility that we can have duplicate data for zip code details of users, i.e. when different users of same city or state register. To overcome this we separate the details of the zip_codes from the Registered_User by creating a separate table called Zip_Codes and we add the zip_code primary key from Zip_Codes as a foreign key to the Registered_User table.

Highest normalization level: 4NF

Table: **Administrator**

Columns:

admin_id (Primary Key)

username

password

Justification: admin_id is unique for every admin user, hence it is chosen as the primary key for Administrator.

Highest normalization level: 4NF

Table: **UserBookFlight**

Columns:

flight_id (Foreign key : references flight_id of Flight)

user_name (Foreign key:references user_name of Registered_User)

no_of_passengers

travel_type

Justification : We have a relationship called "**Registered_User** can book **Flight "** in our conceptual design.To map this to the logical design we are creating a seperate table called **UserBookFlight** by using cross-reference approach.

Highest normalization level: <4NF>

Sprint 3

REQUIREMENTS

General Notes:

- We assume that Guest Users don't need an account as they are just users who can access some basic features without signing up. So we are not storing the data related to Guest User.
- At this time, data about searches conducted by usiers need not be recorded.

Story ID	Story description
US1	As an Administrator, I want to add flight details.
US2	As a User, I want to search for availability of flights.
US3	As a User, I want to check flight schedule.
US4	As a User, I want to check flight fare.
US5	As an Guest user, I want to register so that I can book flight.
US6	As a Registered User, I want to book flight.
US7	As a Registered User, I want to reschedule my flight.
US8	As a Registered User, I want to cancel my flight.
US9	As an Administrator , I want to update the flight details.
US10	As a Registered User, I want to book a flight for round

	trip as well.
US11	As a Registered User, I want to book flight for multiple passengers.
US12	As a User, I want to give a feedback.
US13	As an Administrator, I want to see user feedback.

The role name "User" implies both Guest and Registered Users.

Refined Requirements:

1. As an Administrator, I want to add the flight details so that user can view and search for them.
    a. As a User, I want to perform a search for flight availability between two cities for a given date so that I can check flight schedule and fare

   Notes : We have combined the user stories US2, US3 and US4
2. As a User I want to register so that i can become a Registered User and perform flight booking.
3. As a Registered User , I want to book flight of particular Airline of my choice.

   Notes :
   - Book is generic in practical scenario, hence we incorporate the airline preferences of the users.

5. As an Administrator, I want to update the flight details so that I can update the flight which got delayed or canceled.

6. As an Registered user, I want to reschedule or cancel my flight.

Notes :

In general, rescheduling a flight implies cancelling one flight and booking another. So, we have combined the user stories US7 and US8

NOTES : we are not storing any details regarding the user cancelling or rescheduling the flights and administrator updating the flight details in the database.

CONCEPTUAL DESIGN

Entity: **Flight**

Attributes:

flight_id[Atomic]

flight_number [Atomic]

flight_status [Atomic]

class_type[Atomic]

flight_origin [Atomic]

flight_destination [Atomic]

arrival_time [Atomic]

departure_time [Atomic]

actual_arrival_time [Atomic]

actual_departure_time [Atomic]

flight_date[Atomic]

flight_fare[Atomic]

no_of_seats [Atomic]

flight_delay[derived]

Entity: **Airline**

Attributes:

airline_code [Atomic]

airline_name [Atomic]

Entity: **Registered_User**

Attributes:

user_name [Atomic]

password[Atomic]

Name [composite]

first_name

last_name

email[Atomic]

address [composite]

address_line_1

address_line_2

city

state

zip_code

phone_number[Multi_valued]

credit_card_information[composite]

cc_name

cc_num

cc_expiry

cc_cvv

Entity: **Administrator**

Attributes:

admin_id [Atomic]

username[Atomic]

password[Atomic]

We are assuming Administrator as virtual, so no other details like his name or address are saved in Database.

Action: As an Guest user, I want to register so that I can book a flight.

Relationship: **Registered_User** can book **Flight.**

Cardinality : Many to Many

Participation:

Registered_User has partial participation

Flight has total participation

Relationship: **Flight** belongs to **Airline**

Cardinality : Many to One

Participation:

Flight has total participation

Airline has total participation

Relationship: **Administrator** adds **Flight.**

Cardinality : One to Many

Participation:

Administrator has total participation

Flight has total participation

Relationship: **Registered_User** can feedback for **Flight.**

Cardinality : Many to Many

Participation:

Registered_User has partial participation

Flight has total participation

LOGICAL DESIGN WITH NORMALISATION

Table: **Flight**
Columns:
flight_id (Primary Key)
flight_number

flight_origin

flight_destination

flight_date

flight_status
arrival_time
departure_time
actual_arrival_time
actual_departure_time
airline_code(Foreign key:references airline_code of Airline)

admin_id (Foreign key:references admin_id of Administrator)

Justification :

1. admin_id added as an foreign key to represent the relationship 'Administrator adds flight' (One to Many).
2. airline_code is added as an foreign key to represent the relationship Flight belongs to Airline (Many to One).
3. In an airlines systems, we can have more than one admin as it is a very large database .So we assume that we have more than one admin in our case.

Highest normalization level: 4NF

Indexes for Flight Table

Index type:clustered
Columns:flight_id

Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Flight table which is flight_id and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Index type: non clustered
Columns:airline_code
Justification: This index will be a non clustered index. airline_code is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

Index type: non clustered
Columns:admin_id
Justification: This index will be a non clustered index. admin_id is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

Index type: non clustered
Columns: flight_origin
Justification: This index will be a non clustered index as it will be ordered as the flight_origin of Flight table. Most of the users will search for flights using flight_origin. So it is better to have index for flight_origin as this key will be used by many queries for retrieving records in a table.

Index type: non clustered
Columns:flight_destination
Justification: This index will be a non clustered index as it will be ordered as the flight_destination of Flight table. Most of the users will search for flights using flight_origin. So it is better to have index for flight_destination as this key will be used by many queries for retrieving records in a table.

Table: **Flight_Class**

Columns:

class_id (Primary Key)

class_type

Justification : class_type column distinguishes between the economy class and business class of the flight.

Indexes for Flight_Class Table

Index type:clustered
Columns:class_id
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Flight_Class table which is flight_id and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **Flight_Seating**

Columns:

flight_id (Foreign key: references flight_id of Flight table)

class_id (Foreign key: references class_id of Class table)

no_of_seats

flight_fare

Justification : 'flight_id + class_id' is chosen as Primary key as it uniquely determines the no_of_seats and flight_fare.

Highest normalization level: 4NF

Indexes for Flight_Seating Table

Index type: clustered
Columns: flight_id, class_id
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Flight table which is flight_id and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Index type: non clustered
Columns:flight_id
Justification: This index will be a non clustered index. flight_id is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

Index type: non clustered
Columns:class_id
Justification: This index will be a non clustered index. class_id is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

Table: **Airline**

Columns:

airline_code (Primary Key)

airline_name

Justification:airline_code is unique for every airline, hence it is chosen as the primary key.

Highest normalization level: 4NF

Indexes for Airline Table

Index type: clustered
Columns:airline_code
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Airline table which is airline_code and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **Registered_User**

Columns:

user_name (Primary Key)

password

first_name

last_name

email

address_line_1

address_line_2

zip_code (Foreign key:references zip_code of Zip_Codes table)

phone_number_primary

phone_number_secondary

Justification: user_name is unique for every user, hence it is chosen as the primary key for Registered_User.

Highest normalization level: 4NF

Indexes for Registered_User Table

Index type: clustered
Columns:user_name

Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Registered_User table which is user_name and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **Card_Details**

Columns:

cc_num (Primary Key)

cc_name

cc_expiry

cc_cvv

user_name (Foreign key:references user_name of Registered_User)

Justification:

1. There is a possibility that we can have duplicate data for card details for an user ,i.e.when tickets are booked multiple times by the user.To overcome this we separate the details of the card from the Registered_User by creating a separate table called Card_Details and we add the user_id from Registered_User table as a foreign key to the Card_Details table.

2. A user can use multiple cards for different bookings, hence we username cannot determine any other column. So the table is already in BCNF.

No multivalued dependency between keys, hence the table is finally in 4NF

Highest normalization level: 4NF

Indexes for Card_Details Table

Index type: clustered
Columns:cc_num
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Card_Details table which is cc_num and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **Zip_Codes**
Columns:

zip_code (Primary Key)

city

state

Justification: There is a possibility that we can have duplicate data for zip code details of users, i.e. when different users of same city or state register. To overcome this we separate the details of the zip_codes from the Registered_User by creating a separate table called Zip_Codes and we add the zip_code primary key from Zip_Codes as a foreign key to the Registered_User table.

Highest normalization level: 4NF

Indexes for Zip_Codes Table

Index type: clustered
Columns:zip_code
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Zip_Codes table which is zip_code and it

is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **Administrator**

Columns:

admin_id (Primary Key)

username

password

Justification: admin_id is unique for every admin user, hence it is chosen as the primary key for Administrator.

Highest normalization level: 4NF

Indexes for Administrator Table

Index type: clustered
Columns:admin_id
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Administrator table which is admin_id and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Table: **UserBookFlight**

Columns:

user_name (Foreign key:references user_name of Registered_User)

flight_id (Foreign key : references flight_id of Flight)

no_of_passengers

travel_type

Justification : We have a relationship called "**Registered_User** can book **Flight "** in our conceptual design.To map this to the logical design we are creating a seperate table called **UserBookFlight** by using cross-reference approach.

Highest normalization level: <4NF>

Indexes for UserBookFlight Table

Index type: clustered
Columns:user_name, flight_id
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the UserBookFlight table which is user_name and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Index type: non clustered
Columns:user_name
Justification: This index will be a non clustered index. user_name is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

Table: **UserFeedback**

user_name (Foreign key:references user_name of Registered_User)

flight_id (Foreign key:references user_name of Registered_User)

comments

Justification : We have a relationship called "**Registered_User** can give feedback for **Flight"** in our conceptual design.To map this to the logical design ,we are creating a seperate table called **UserFeedback** by using cross-reference approach**.**

Highest normalization level: <4NF>

Indexes for UserFeedback Table

Index type: Clustered
Columns:user_name ,flight_id
Justification: This index will be a clustered index as it will be ordered in the same way as the primary key of the Registered_User table which is user_name and it is usually a good practice to have index for primary key as this key will be used by many queries for retrieving records in a table.

Index type: Non Clustered
Columns:user_name
Justification: This index will be a non clustered index. user_name is a foreign key. It makes a lot of sense to index all the columns that are part of any foreign key relationship, because through a Foreign Key-relationship, we will often need to lookup a relating table and extract certain rows based on a single value or a range of values. So it makes good sense to index any columns involved in a Foreign Key.

VIEWS AND STORED PROGRAMS

View : UserFeedbackView
Goal: This view is used to see the feedbacks given by users for all flights and only admin can see feeback.

View : customerInfo
Goal : This view is used to see the information of all users like their names,emails and addresses. This view can be used by the Airline systems to know about their customers.

View :SearchForFlight
Goal : User can check for the availability of flights and its details like fare ,schedule etc..

Stored procedure: userFeedback

Parameter	Parameter Type
userName	IN
flightId	IN
commentsUser	IN

Goal:
This procedure is used to keep a log of the comments/reviews that passengers provide for the flight they had travelled.

Stored procedure: updateFlightStatus

Parameter	Parameter Type
flightId	IN

flightstatus	IN
newArrivalTime	IN
newDepartureTime	IN

Goal:
This procedure is used to update the status of the flight (OnTime/Cancelled/Delayed).
For the flight status as Delayed for a particular flight, the latest arrival and departure time of the delayed flight is updated along with the status.

Stored procedure: userRegistrationProc

Parameter	Parameter Type
userName	IN
Pwd	IN
firstName	IN
lastName	IN
Email	IN
addressLine1	IN
addressLine2	IN
primaryNumber	IN
secondaryNumber	IN
zipCode	IN

Goal:

This procedure is used to register the user so that he/she can book the flight.

Stored procedure: userBookFlightProc

Parameter	Parameter Type
userName	IN
flightId	IN
numOfPassengers	IN
travelType	IN

Goal:
This procedure is used to store booking details of flight according to user inputs.

Stored procedure: AdminaddsFlight

Parameter	Parameter Type
flightno	IN
flightstatus	IN
flightorigin	IN
flightdest	IN
arrtime	IN
deptime	IN
actarr_time	IN
actdep_time	IN
flightdate	IN

airlinecode	IN
adminid	IN
classid	IN
fare	IN
seats	IN
classid1	IN
fare1	IN
seats1	IN

Goal:
This procedure is used to add flights by the admin so when the user searches for availability of flight he can check and book.