**Project team #8 –** **PlanMyTravel**

**Team Members:**

Sanju Kurubara Budi Hall Hiriyanna Gowda

Yousuf sadiq mohammed

Vipul Shukla

Sai Koumudi Kaluvakolanu

## **Project Proposal:**

**Overview and Motivation**

The idea behind the application is to provide a platform for searching, selecting and booking the flights based on the availability of the flights. The application will provide an easy access to various flight belonging to different airlines without the hassle of checking the flights individually. User will be able to plan his travel in advance by comparing the prices of the flights.

**Scope and Objectives**

* PlanMyTravel will provide a web based interface for users to search and book flights based on the availability.
* User must be able to signup/sign in into the application using their email Id and can make the flight bookings.
* Users should be able to choose the source, destination, date of journey, choice of meal and the class they wish to travel in.
* User will have an option to book a one-way or a round trip journey.
* User can select one from all the displayed results and should be able to book it successfully.
* User should be able to view the available seats and book accordingly. User can make a payment either thru credit/debit card.
* If the User wishes to store the card details for further transactions, it can be done. Payment should be done successfully.
* A confirmation mail should be sent to the respective mail Id with the booking reference number.
* Also, a reminder mail will be sent on the day before the journey.
* User should be able to reschedule or cancel the booked flight.
* After the travel, the User can also rate his/her journey experience and provide feedback.

The intended users of this application will be any person who might want to book flights and travel from one place to another.

**Environment setup:**

Eclipse is being used for designing simple UI, establishing the server (Apache Tomcat) connections and integrating with mySQL. The programming languages being used are HTML, Java, Servlets, JSP.

**High Level User Requirements:**

**User Roles:**

● Guest user

-Able to choose source, destination, date of travel, one-way/round trip and can view the flight details accordingly.

● Registered user

-Able to login to the Application.

-Able to choose source, destination, date of travel, one-way/round trip and can view the flight details accordingly.

-Able to book the selected flight by making payment and by choosing a meal plan.

-Able to update profile details.

-Able to save the card details for future transactions.

-Able to view the booked flight details.

-Able to cancel or reschedule the booked flight.

-Able to rate the journey.

-Able to check-in.

-Able to modify the meal plan.

● Admin

-Able to login to the Application.

-Able to manage the flight details and meal plan.

**Prioritizing the User Stories in Decreasing order:**

|  |  |  |
| --- | --- | --- |
| **User Roles** | **User Stories** | **Priority** |
| Admin | As an Admin, I want to login to PlanMyTravel | 1 |
| Admin | As an Admin, I want to add the details of new itinerary | 2 |
| Admin | As an Admin, I want to delete the existing itinerary | 3 |
| Guest User | As a Guest user, I want to search the availability of flights based on source and destination on a particular date | 4 |
| Guest User | As a Guest user, I want to signup on PlanMyTravel | 5 |
| Registered user | As a Registered user, I want to login on PlanMyTravel | 6 |
| Registered user | As a Registered user, I want to search the availability of flights based on source and destination on a particular date | 7 |
| Registered user | As a Guest user, I want an option to select either one-way or round trip booking | 8 |
| Registered user | As a Registered user, I want an option to select either one-way or round trip booking | 9 |
| Registered user | As a Registered user, I want to select a flight from the list of available flights | 10 |
| Registered user | As a Registered user, I want to book the selected flight | 11 |
| Registered user | As a Registered user, I want to make a payment through my credit/debit card | 12 |
| Registered user | As a Registered user, I want to receive the booking reference number to my registered mail | 13 |
| Registered user | As a Registered user, I want to receive the itinerary details to my registered mail | 14 |
| Registered user | As a Registered user, I want to receive a notification on my registered mail one day prior to my date of journey | 15 |
| Registered user | As a Registered user, I want an option to reschedule my flight booking | 16 |
| Registered user | As a Registered user, I want an option to cancel my flight booking | 17 |
| Registered user | As a Registered user, I want an option to save the card details so that I can use them for future transactions | 18 |
| Admin | As an Admin, I want to update the flight schedule | 19 |
| Admin | As an Admin, I want to add a meal plan | 20 |
| Admin | As an Admin, I want to delete a meal plan | 21 |
| Registered user | As a Registered user, I want an option to select a meal for my travel | 22 |
| Registered user | As a Registered user, I want an option to perform online check-in | 23 |
| Registered user | As a Registered user, I want an option to rate my travel | 24 |

**High Level Conceptual Design:**

**Entities:**

● registered\_user

● admin

● flight\_details

● booking\_details

● meal\_plan

● card\_details

**Relationships:**

● admin manages the flight\_details

● admin manages the meal\_plan

● registered\_user manages booking\_details

**Actions:**

* registered\_user can check the flight\_details
* registered\_user can save the card\_details
* registered\_user modifies meal\_plan

**SPRINT 1:**

**User stories in decreasing order of priority:**

1. As an Admin, I want to login to PlanMyTravel.
2. As an Admin, I want to add the details of new itinerary.
3. As a Guest user, I want an option to choose source and destination of flight.

Notes:

* Source and destination should be a valid place

1. As a Guest user, I want an option to choose the date of journey

Notes:

* Date of journey should be valid

1. As a Guest user, I want an option to search the availability of flights

Notes:

* Search is based on user’s input on source, destination, date of journey

1. As a Guest user, I want to signup on PlanMyTravel
2. As a Registered user, I want to login on PlanMyTravel

**Conceptual Design:**

Entity: **registered\_user**

Attributes:

user\_name[primary key]

password

name [multivalued]

first\_name

last\_name

email\_id

Entity: **admin**

Attributes:

user\_name[primary key]

password

name [multivalued]

first\_name

last\_name

`email\_id

Entity:**flight\_details**

Attributes:

flight\_id[primary key]

flight\_number

source\_id

destination\_id

arrival\_time

departure\_time

date\_of\_arrival

date\_of\_departure

economy\_seat\_count

business\_seat\_count

Entity: **flight**

Attributes:

flight\_number[primary key]

airlines\_id[foreign key references airlines\_id in Airlines table]

Entitiy**: cities**

Attributes**:**

city\_id[primary key]

ity\_name

country\_id

Entity**: airlines**

Attributes**:**

airlines\_id[primary Key]

airlines\_name

**Relationships:**

**1. admin** adds **flight\_details**

**Cardinality: one to many**

**Participation: Admin has partial participation**

**flight\_details has total participation.**

**Justification:** Admin has partial participation because there will be one admin working on adding flight details.

**2.** **registered\_user** searches **flight\_details**

**Cardinality: many to many**

**Participation: registered\_user has partial participation**

**flight\_details has partial participation.**

**Actions:**

**Guest\_user** sign ups on the **website.**

**Registered\_user** logins into the **application.**

**Logical Design:**

Table: **registered\_user**

Columns:

user\_name[primary key]

password

first\_name

last\_name

email\_id

**Justification**: This table includes users who have registered on plan my travel website. Each user is uniquely identified by its user\_name. Also, name is a composite attribute so, we have separated it into two columns.

**Highest normalization level: 2 NF**

**Justification**: Using email\_id, we can extract first\_name, last\_name and user\_name. However, we don't want to separate the user details into another table since it requires more join operation to retrieve data which in turn degrades the performance.

Table: **admin**

Columns:

username[primary key]

password

first\_name

last\_name

email\_id

**Justification**: This table includes admins who can add/delete/update flight plans on the website.

**Highest normalization level: 2 NF**

**Justification**: Using email\_id, we can extract first\_name, last\_name and user\_name. However, we don't want to separate the user details into another table since it requires more join operation to retrieve data which in turn degrades the performance.

Table: **flight\_details**

Columns:

flight\_id[primary key]

flight\_number[foreign key references flight\_number in flight table]

source\_id[foreign key references city\_id in cities table ]

destination\_id[foreign key references city\_id in cities table ]

arrival\_time

departure\_time

date\_of\_arrival

date\_of\_departure

economy\_seat\_count

business\_seat\_count

Justification: flight\_details includes various operating flights by different airlines. Each flight is uniquely identified by its flight\_id.

**Highest normalization level: 4 NF**

Table: **flight**

Columns:

flight\_number[primary key]

airlines\_id[foreign key references airlines\_id in Airlines table]

**Highest normalization level: 4 NF**

**Table: cities**

**Columns:**

city\_id[primary key]

city\_name

Justification: To identify each row uniquely we have added city\_id as a column based on the auto increment feature. Each city is identified by a specific city\_id.

**Highest normalization level: 4 NF**

**Table: Airlines**

**Columns:**

airlines\_id[primary Key]

airlines\_name

Justification: This table consists of different airlines using this website. To identify each row uniquely we have added airlines\_id as a column based on the auto increment feature. Each airlines has a unique airlines\_id.

**Highest normalization level: 4 NF**

**SPRINT 2:**

**User stories in decreasing order of priority:**

1. As a Registered user, I want to select a flight from the list of available flights
2. As an Admin, I want to add a meal plan
3. As a Registered user, I want to book the selected flight
4. As a Registered user, I want to make a payment through my credit/debit card

* An option to select either credit or debit card
* A field to enter the name on the card, card number, expiry date, cvv and type of card (as in visa, master)

1. As a Registered user, I want an option to save the card details so that I can use them for future transactions
2. As a Registered user, I want an option to select a meal for my travel
3. As a Registered user, I want an option to reschedule my flight booking
4. As an Admin, I want to delete the existing itinerary

**Conceptual Design:**

Entity:**meal\_plan**

Attributes:

meal\_id[primary key]

meal\_name

**Relationships:**

**1.** **registered\_user** books **flight\_details**

**Cardinality:** many to many

**Participation:** registered\_user has partial participation

flight\_detailshas partial participation.

**2. admin** adds **meal\_plan**

**Cardinality:** one to many

**Participation:** admin has partial participation

meal\_plan has total participation

**Action:**

**registered\_user** saves **card\_details**

**Cardinality:** one to many

**Participation:** registered\_user has partial participation

card\_details have total participation

**Logical Design:**

Table:**meal\_plan**

Columns:

meal\_id[primary key]

meal\_name

**Highest normalization level: 4 NF**

Table:**booked\_flights**

Columns:

booking\_id[primary key]

user\_name[foreign key references user\_name in registered\_user]

flight\_id[foreign key references flight\_id in flight\_details]

no\_of\_economy\_seats[default 0]

no\_of\_business\_seats[default 0]

checkin\_flag[default 0]

feedback\_flag[default 0]

**Justification:**Using cross reference approach to represent the many to many relationship “**registered\_user** books **flight\_details**”. We are creating a separate table called **booked\_flights,** to store the booking details.

**Highest normalization level: 4 NF**

Table: **passenger\_details**

Columns:

passport\_no

booking\_id[foreign key references booking\_id in booked\_flights table]

passenger\_last\_name

passenger\_first\_name

meal\_id[foreign key references meal\_id in meal\_plan,default 1]

**Highest normalization level: 4 NF**

Table: **card\_details:**

Columns:

card\_id[primary key]

user\_name[foreign key references user\_name in registered\_user table]

card\_no

name\_on\_card

cvv

card\_type

Payment\_method

expiry\_date

**Justification**: Using foreign key approach to represent the one to many relationship “**registered\_user** saves **card\_details**” because it is higher overhead to create a separate table to represent this relationship.

**Highest normalization level: 2 NF**

**Justification:** Using card\_no, we can extract cvv, name\_on\_card and card\_type. However, we don't want to separate the card details into another table since it requires more join operation to retrieve data which in turn degrades the performance.

**SPRINT 3:**

**User stories in decreasing order of priority:**

1) As an Admin, I want to update the flight schedule.

2) As an Admin, I want to delete a meal plan.

3) As a Registered user, I want an option to cancel my flight booking.

4) As a Registered user, I want an option to perform online check-in.

5) As a Registered user, I want an option to rate my travel.

6) As a Registered user, I want to receive the booking reference number to my registered mail.

7) As a Registered user, I want to receive the itinerary details to my registered mail.

8) As a Registered user, I want to receive a notification on my registered mail one day prior to my date of journey

**High Level Conceptual Design:**

**Action:**

* **registered\_user** gives **feedback on booked\_flights**

**Logical Design:**

Table: **registered\_user**

Columns:

user\_name[primary key]

password

first\_name

last\_name

email\_id

**Justification**: This table includes users who have registered on plan my travel website. Each user is uniquely identified by its user\_name. Also, name is a composite attribute so, we have separated it into two columns.

**Highest normalization level: 2 NF**

**Justification**: Using email\_id, we can extract first\_name, last\_name and user\_name. However, we don't want to separate the user details into another table since it requires more join operation to retrieve data which in turn degrades the performance.

**Index:** Clustered

**Columns:** user\_name

**Justification:** The registered\_user table is queried very often using user\_name. Since, user\_name is the primary key, we can make it as a clustered index. For example, if we want to retrieve the user details, we’ll perform search operation based on user\_name.

**Index:** Non-Clustered

**Columns:** email\_id

**Justification:**The registered\_user table is also queried very often using email\_id. So, we can make it as a non-clustered index. For instance, we may want to search for user details including user\_name, we can do it using email\_id.

Table: **admin**

Columns:

username[primary key]

password

first\_name

last\_name

email\_id

**Justification**: This table includes admins who can add/delete/update flight plans on the website.

**Highest normalization level: 2 NF**

**Justification**: Using email\_id, we can extract first\_name, last\_name and user\_name. However, we don't want to separate the user details into another table since it requires more join operation to retrieve data which in turn degrades the performance.

**Index:** Clustered

**Columns:** username

**Justification:** The admin table is queried very often using user\_name. Since, user\_name is the primary key, we can make it as a clustered index.For example, if we want to retrieve the admin details, we’ll perform search operation based on username.

**Index:** Non-Clustered

**Columns:** email\_id

**Justification:**The admin table is also queried very often using email\_id. So, we can make it as a non-clustered index. For instance, we may want to search for admin details including username, we can do it using email\_id.

Table: **flight\_details**

Columns:

flight\_id[primary key]

flight\_number[foreign key references flight\_number in flight\_info table]

source\_id[foreign key references city\_id in cities table ]

destination\_id[foreign key references city\_id in cities table ]

arrival\_time

departure\_timel

date\_of\_arrival

date\_of\_departure

economy\_seat\_count

business\_seat\_count

Justification: flight\_details includes various operating flights by different airlines. Each flight is uniquely identified by its flight\_id.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** flight\_id

**Justification:**The flight\_details table is can be queried using flight\_id. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** source\_id, destination\_id, flight\_number

**Justification:**The flight\_details table is queried very often using both source\_id, destination\_id and flight\_number. So, we can make it as a non-clustered index. It is quite obvious that the search operation on flight\_details is based on source\_id, destination\_id and flight\_number.

Table: **flight\_info**

Columns:

flight\_number[primary key]

airlines\_id[foreign key references airlines\_id in Airlines table]

total\_economy\_seats

total\_business\_seats

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** flight\_number

**Justification:** The flight\_info table is queried very often using flight\_number. Since, flight\_number is the primary key, we can make it as a clustered index. For instance, to know the flight details, like to which airlines it belongs, how many seats are there in economy and business class, we need to search based on flight\_number.

**Index:** Non-Clustered

**Columns:** airlines\_id

**Justification:** The flight\_info table is queried very often using airlines\_id and can be considered as non clustered index.

**Table: cities**

**Columns:**

city\_id[primary key]

city\_name

country\_id[foreign key references country\_id in country table]

Justification: To identify each row uniquely we have added city\_id as a column based on the auto increment feature. Each city is identified by a specific city\_id.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** city\_id

**Justification:**The cities table is can be queried using city\_id. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** city\_name

**Justification:**The cities table is queried very often using city\_name. So, we can make it as a non-clustered index. To know, if the flight exists from a particular city, we’ll query frequently using the city\_name.

**Table**: **country**

**Columns**:

country\_id[primary key]

country\_name

Justification: To identify each row uniquely we have added country\_id as a column based on the auto increment feature. Each country is identified by a specific country\_id.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** country\_id

**Justification:**The country table can be queried using country\_id. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** country\_nameinsett

**Justification:**The country table is queried very often using country\_name. So, we can make it as a non-clustered index. For example, to know how many flights exist from a particular country, we perform search based on country\_name.

**Table: airlines**

**Columns:**

airlines\_id[primary Key]

airlines\_name

Justification: This table consists of different airlines using this website. To identify each row uniquely we have added airlines\_id as a column based on the auto increment feature. Each airlines has a unique airlines\_id.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** airlines\_id

**Justification:**The airlines table can be queried using airlines\_id. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** airlines\_name

**Justification:**The airlines table is queried very often using airlines\_name. So, we can make it as a non-clustered index. we use airlines\_name to know which airlines are running frequently.

Table:**meal\_plan**

Columns:

meal\_id[primary key]

meal\_name

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** meal\_id

**Justification:**The meal\_plan table can be queried using meal\_id. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** meal\_name

**Justification:**The meal\_plan table is queried very often using meal\_name. So, we can make it as a non-clustered index. For example, anyone wishes to search their meal\_plan based on meal\_name.

Table:**booked\_flights**

Columns:

booking\_id[primary key]

user\_name[foreign key references user\_name in registered\_user]

flight\_id[foreign key references flight\_id in flight\_details]

no\_of\_economy\_seats[default 0]

no\_of\_business\_seats[default 0]

checkin\_flag[default 0]

feedback\_flag[default 0]

Seats\_booked[default 0]

**Justification:**Using cross reference approach to represent the many to many relationship “**registered\_user** books **flight\_details**”. We are creating a separate table called **booked\_flights,** to store the booking details.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** booking\_id

**Justification:**The booked\_flights table is queried very often using booking\_id which is a primary key. So, we can make it as a clustered index. For instance, to determine the flight details that are booked, booking\_id will be used commonly.

**Index:** Non-Clustered

**Columns:** flight\_id, user\_name

**Justification:**The booked\_flights table is queried very often using flight\_id and user\_name. So, we can make it as a non-clustered index. For example, there are people who prefer to know the flight details based on flight\_id/user\_name and so, it is made as non-clustered index.

Table: **passenger\_details**

Columns:

passport\_no[primary key]

booking\_id[foreign key references booking\_id in booked\_flights table]

passenger\_last\_name

passenger\_first\_name

meal\_id[foreign key references meal\_id in meal\_plan,default 1]

Justification: Combination of passport\_no and booking\_id is the primary key.

**Highest normalization level: 2 NF**

**Justification:**Using email\_id, we can extract first\_name, last\_name and user\_name. However, we don't want to separate the user details into another table since it requires more join operation to retrieve data which in turn degrades the performance

**Index:** Clustered

**Columns:** passport\_no

**Justification:**The **passenger\_details** table can be queried using passport\_no. So, we can make it as a clustered index.

**Index:** Non-Clustered

**Columns:** booking\_id

**Justification:**The passenger\_details table is queried very often using booking\_id. So, we can make it as a non-clustered index. The booking\_id uniquely determines the passenger details and is used frequently.

Table: **card\_details:**

Columns:

user\_name[foreign key references user\_name in registered\_user table]

card\_no

name\_on\_card

cvv

card\_type

Payment\_method

expiry\_date

**Justification**: Using foreign key approach to represent the one to many relationship “**registered\_user** saves **card\_details**” because it is higher overhead to create a separate table to represent this relationship.

**Highest normalization level: 4 NF**

**Index:** Clustered

**Columns:** user\_name, card\_no

**Justification:**The user\_card\_details table is queried very often using user\_name and card\_no. So, we can make it as a clustered index as it is a primary key in the table. Whenever, we need to have the card details of a particular user, we can retrieve them using the user\_name.

**Stored programs:**

**Stored procedure:** insert\_passenger\_details

**Parameters:**IN id(This is the booking id)

IN passenger\_passport(This is the list of passport numbers of the passengers)

IN passenger\_meal(These are the list of meal id’s corresponding to passport numbers)

IN passenger\_last\_name(These are the list of passengers last names corresponding to their passport numbers),IN passenger\_first\_name (These are the list of passengers first names corresponding to their passport numbers)

**Goal:** This procedure takes passenger\_details as input and insert values into the passenger\_details table for its corresponding booking id.

**Stored procedure:** book\_flight

**Parameters:**IN f\_id(This is the flight\_id)

IN user\_name(This is the name of the user who books the flight)

IN economy(This is the number of seats in the economy class)

IN business(This is the number of seats in the business class)

IN passenger\_passport(This is the list of passport numbers of the passengers)

IN passenger\_meal(These are the list of meal id’s corresponding to passport numbers)

IN passenger\_last\_name(These are the list of passengers last names corresponding to their passport numbers)

IN passenger\_first\_name (These are the list of passengers first names corresponding to their passport numbers)

**Goal:** This procedure checks if the number of seats are greater than the available seats if they are greater then it inserts data into booked\_flights table and later calls insert\_passenger\_details procedure where the list of passengers are stored in the passenger\_detailstable and seat count is deducted in the flight\_details table based on the number of seats chosen by the user.

**Stored procedure:** insert\_card\_details

**Parameters:**IN card\_no (This is the card number)

IN name\_on\_card(This is the name of user on the card)

IN cvv(CVV of the card)

IN card\_type(Card type is either visa or master)

IN user\_name(User name who has stored the card details)

IN payment\_method(It is either credit or debit)

IN expiry\_date(Card expiry date)

**Goal:** This procedure saves the card details of the user into card\_details table

**Stored procedure:** delete\_booking\_details

**Parameters:**IN id (This is the booking id)

**Goal:** This procedure deletes the booking details of the user from both the passenger\_details table and booked\_flights table.

**Stored procedure:** delete\_flight\_details

**Parameters:**IN id (This is the flight id)

**Goal:** This procedure deletes the booking details of the user from both the passenger\_details table and booked\_flights table and later it deletes from the flight\_details table.

**Stored procedure:** delete\_meal\_plan

**Parameters:**IN name1 (This is the meal name)

**Goal:** This procedure deletes the row corresponding to the meal name.

**Stored procedure:** insert\_meal\_plan

**Parameters:**IN name1 (This is the meal name)

**Goal:** This procedure inserts the row corresponding to the meal name.

**Stored procedure:** insert\_admin

**Parameters:**IN user1(This is the username of the admin)

IN pass(This is the password of the admi)

IN email\_id(This is the email id of the admin)

IN first1(This is the first name of the admin)

IN last1(This is the last name of the admin)

**Goal:** This procedure inserts the row into the admin table for a new admin.

**Stored procedure:** insert\_registered\_user

**Parameters:**IN user1(This is the username of the admin)

IN pass(This is the password of the admi)

IN email\_id(This is the email id of the admin)

IN first1(This is the first name of the admin)

IN last1(This is the last name of the admin)

**Goal:** This procedure inserts the row into the registered\_user table when a new user is registered.

**Stored procedure:** insert\_booking\_details

**Parameters:**IN user\_name(This is the username of the user who has booked the flight)

IN flight\_id(This is the flight\_id of the flight booked by the user)

IN economy(This is the number of economy seats booked by the user)

IN business(This is the number of business seats booked by the user)

**Goal:** This procedure inserts the row into the booked\_details table whenever a user books a flight.

**Stored procedure:** insert\_flight\_details

**Parameters:**IN number1(This is the flight number of the itinerary added)

IN a\_time(This is the arrival time of the flight)

IN d\_time(This is the departure time of the flight)

IN d\_arrival(This is the date of arrival of the flight)

IN d\_departure(This is the date of departure of the flight)

IN economy(This is available economy class seats in the flight)

IN business(This is available business class seats in the flight)

IN sourcename(This is the name of the source city)

IN destname(This is the name of the destination city)

**Goal:** This procedure gets the source and destination id’s from the cities table inserts the row into the fligth\_details table.

**Functions:**

**Stored function: check\_in**

**Parameters:**IN id(This is the booking id of the user for check in option)

**Goal:** This procedure checks if the checkin\_flag is ‘0’ for the corresponding booking\_id and if the zero then it updates the checkin\_flag to ‘1’ and returns ‘1’, else it returns zero.

**Stored function: feedback**

**Parameters:**IN id(This is the booking id of the user for feedback option)

**Goal:** This procedure checks if the feedback\_flag is ‘0’ for the corresponding booking\_id and if the zero then it updates the feedback\_flag to corresponding input and returns ‘1’, else it returns zero.

**Event:**

Created an event to automatically create a row 1 day prior to the date of journey in email\_notification table. Same table will be used to send the notification email to the respective emailid.

**Below is the example for the same**

CREATE EVENT IF NOT EXISTS event\_notification\_(booking\_id)\_(username)

ON SCHEDULE AT (date of booking) - interval 1 day

DO

INSERT INTO email\_notification(booking\_id,email)

VALUES('last\_booking\_id',email\_id);