## horizontal line

Btech ICT - Sem 4

**Database Management System**

Railway - Management - System

**Group Members:**

AU1841005 Yashil Depani

AU1841007 Ridham Shah

AU1841010 Harshil Mehta

# Description

Starting off, we have tried to model a mini online railway reservation system. To do so, we have used PostgreSQL for database design and query mode and other back - end activity. We have used python language as our medium , html web pages for frontend and flask for connecting frontend with postgresql.

For booking tickets, a user must log in to the system or create a new account and get his user ID which is used to book tickets and further reservation purposes through a Login Module.

For the process of Booking tickets, a schedule of trains is shown to users with appropriate filters. Schedule of the train includes train details with timings, stations,seats left and ticket price. There are many types of seat Category and if booked on any specific train and specific date, a unique ‘pnr’ is generated for that booking. Also, passengers are queued, if there are no more seats available then this passenger is added to the ‘waiting’ queue. The passenger also has the ‘cancel’ ticket feature. If any passenger cancels the ticket, the queue is pushed and the front of the queue is booked. Meaning the passenger got his ticket status changed from ‘waiting’ to ‘confirmed’. The payment is processed through the user’s wallet. The minimum balance of ticket price needs to be there in the user’s wallet.

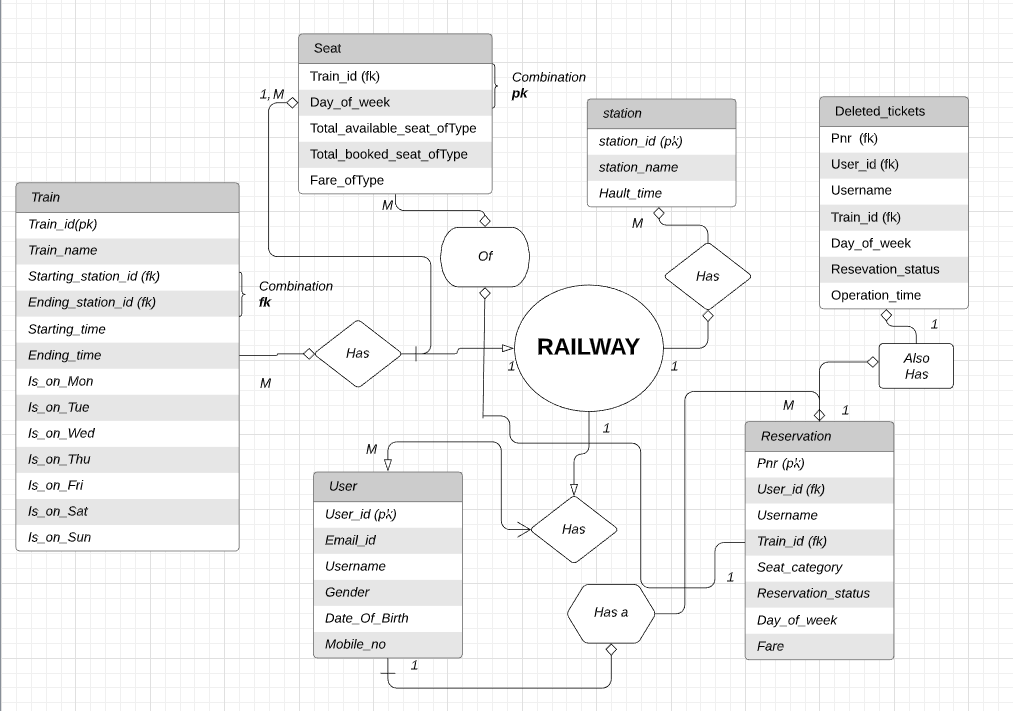
Admin side Functionalities include Inquiry functions such as Inquiry for trains traveling from point A to point B, details of all trains traveling between two dates, details of any station, the revenue generated on the given date for a train, details of deleted tickets, and inquiries like booking status and other formal inquiries.

In addition to other features, we’ve also included the technical aspect of deletion of any records by any user/administrator which in return deletes all child records via triggers and functions/procedures. We’ve also created a record for tracking all the user-driven activities. Input details are validated for safe data handling.

We also have our special function and we are happy that our team was able to implement that. Say a user wants to go from station A to B, have some time to spend at Station B because of his work and go from station B to C. So the user has the flexibility to avail such special options and appropriate output is displayed.

All these functionalities are implemented with appropriate procedures/functions and triggers.

# Entity - Relational Diagram



This shows the inter - dependency of the entities on each other with respect to their attributes.

# Table Design (Data - Dependency)

* Train Table *(Entity)*
  + **ATTRIBUTES:**
  + Train\_id int: Is an assigned **primary key** to uniquely identify any train record and its details.
  + Train\_name varchar(20): Name of any specific train.
  + Starting\_station\_id int: Is a **foreign key** that is referenced from the station table to specify the starting station of any train.
  + Ending\_station\_id int: Is a **foreign key** that is referenced from the station table to specify the end station of any train.
  + Start\_time float: Used to know about the departure time of any train in format HH.MM where HH = hours, MM = minutes.
  + Journey\_time float: For the total journey of the particular train.
  + Mon, Tue, Wed, Thu, Fri, Sat, Sun char: All are used to specify the days of the week when the train is active.
* User Table (*Entity*)
  + **ATTRIBUTES:**
  + User\_id int: Is an assigned **primary key** to uniquely identify each user.
  + Email\_id varchar(50): Email id of the user.
  + Name varchar(30): Name of the particular user.
  + Gender char, dob(date of birth) date, mobile\_no int.
* Reservation Table (*Entity*)
  + **ATTRIBUTES:**
  + Pnr serial int: Is an assigned **primary key** that uniquely identifies each reservation made.
  + User\_id int: Used as **foreign key** from the user table that determines which user has made a reservation.
  + Username varchar(30): Used to determine the name of the user who made the reservation.
  + Trai
  + n\_id int: Used as a **foreign key** in determining on which train was the reservation made.
  + Day\_of\_week char: To identify on which day the booking was made.
  + Reservation\_status char: Used to know the booking status of the reservation made.
  + Seat\_category varchar(3): Used to determine the category of the reserved seat.
  + Fare int: The price of the reserved seat.
* Station Table (*Entity*)
  + **ATTRIBUTES:**
  + Station\_id int: Is an assigned **primary key** that uniquely identifies each station.
  + Station\_name varchar(20): Name of station.
  + Halt\_time float: Halt time at the particular station in MM.SS (minutes.seconds) format.
* Seat table (*Entity*)
  + **ATTRIBUTES:**
  + Train\_id int: Is a **foreign key** which indicates that to which train does the seat belong.
  + Day\_of\_week char: Used to know the day on which the train is to depart.
  + The combination of train\_id and day\_of\_week is the **primary key** for the given table.
  + Total\_booked seat of types(below): For the booked seat of a given type.
    - AC1\_book int
    - AC2\_book int
    - AC3\_book int
    - CC\_book int
    - EC\_book int
    - SL\_book int
  + Total\_available seat of types(below)**:** For the number of total available seats of a given type.
    - AC1\_total int
    - AC2\_total int
    - AC3\_total int
    - CC\_total int
    - EC\_total int
    - SL\_total int
  + Fare of the seat of types(below): To determine the rate of the seat.
    - AC1\_price int
    - AC2\_price int
    - AC3\_price int
    - CC\_price int
    - EC\_price int
    - SL\_price int
* Deleted\_tickets (*Entity*)[For keeping track of record]
  + **ATTRIBUTES:**
  + Pnr serial int: Used as a **foreign key** for determining the pnr of canceled reservation.
  + User\_id int: Used as a **foreign key** for determining the user id of canceled Passenger.
  + Username varchar(30): Used for determining the user name of Passenger who canceled the booking.
  + Train\_id int: Used as a **foreign key** for determining the train in which the passenger canceled the booking.
  + Day\_of\_week char: Used to know the day on which the train was set to depart.
  + Reservation\_status char: Used to determine whether the booking was confirmed or not.
  + Operation\_time timestamp: Used to keep track of time when the cancellation was performed.

**Interdependencies**

The **Train table** has different attributes as described above with its nature(primary key, foreign key, etc). It has it’s starting station and its ending station referenced from the station table to use it’s station id and other attributes. This table mainly connects a particular train running on respectives days with stations.

The **Seat table** signifies the normalized form of E-R diagram as it contains different types of seats and prices respectively.Also, it has referenced train\_id to know which train does this seat belong to. The user through this can know the price and its category of seat with remaining count in order to avail a ticket.

The **User table** has different details entaling any user which signs up for this system. It only contains the essential details of a customer like name,password,mobile number, etc with its wallet details.

The **Reservation table** has different attributes with a unique PNR serial and has referenced attributes such as user\_id and train\_id to know the details of the booking made.It also has the reservation status and the category of seat for which the booking is made. This table also keeps user status such as confirmation or waiting

The **Station table** has different attributes which are used to know the details of any given station.

The **deleted\_tickets table** is a kind of track table which keeps the track of all cancelled bookings by user by timestamps.The different attributes it has are user\_id, PNR, train\_id to identify each booking uniquely. Basically it is a table to keep records of who deleted tickets on which date , on which train etc.

These are connected in the above shown 3NF’ed E-R diagram with proper relationship between tables.

# Stored Procedures and Functions Codes:

**Functions:**

**Function - 1**: Function to display all the trains which are running from station x to station y on a particular date where x and y are inputs. It represents a whole table.

Code:

--procedure for schedule: input is from station id and to station id , date

--output is table

create or replace function dis\_train2(s\_stat int, e\_stat int, tarik date)

returns table(

s\_stat\_name varchar(20),

e\_stat\_name varchar(20),

t\_name varchar(20),

t\_id int,

s\_time float,

j\_time float,

ac1\_p int,

ac1\_s int,

ac2\_p int,

ac2\_s int,

ac3\_p int,

ac3\_s int,

cc\_p int,

cc\_s int,

ec\_p int,

ec\_s int,

sl\_p int,

sl\_s int

)

as $dis\_train2$

declare

dayno int;

begin

if(tarik<current\_date) then

raise exception 'Date invalid';

end if;

select extract(dow from tarik) into dayno;

return query

select distinct

(select station\_name from station1 where station\_id=s\_stat),

(select station\_name from station1 where station\_id=e\_stat),

train1.train\_name,train1.train\_id,train1.starting\_time,train1.journey\_time,

seat\_class2.ac1\_price,seat\_class2.ac1\_total\_seats-seat\_class2.ac1\_booked\_seats,

seat\_class2.ac2\_price,seat\_class2.ac2\_total\_seats-seat\_class2.ac2\_booked\_seats,

seat\_class2.ac3\_price,seat\_class2.ac3\_total\_seats-seat\_class2.ac3\_booked\_seats,

seat\_class2.cc\_price,seat\_class2.cc\_total\_seats-seat\_class2.cc\_booked\_seats,

seat\_class2.ec\_price,seat\_class2.ec\_total\_seats-seat\_class2.ec\_booked\_seats,

seat\_class2.sl\_price,seat\_class2.sl\_total\_seats-seat\_class2.sl\_booked\_seats

from train1,seat\_class2,station1

where(train1.starting\_station\_id=s\_stat and train1.ending\_station\_id=e\_stat and train1.train\_id=seat\_class2.train\_id

and seat\_class2.working\_day=dayno);

end;

$dis\_train2$ language plpgsql;

--How to call?

select dis\_train2(100,101,'20-04-2020')

**Function - 2**: Function to book a seat in any given train and given seat category on any given day.

Code:

--function for booking a seat

select book\_seat(10000,1,'AC1')

create or replace function book\_seat(t\_id int, days int, categ varchar)

returns int as $book\_seat$

declare

exist int;

left int;

begin

if(categ='AC1') then

select ac1\_total\_seats,ac1\_booked\_seats into exist,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice 'No seats left';

return ac1\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set ac1\_booked\_seats=ac1\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return ac1\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

end if;

elsif(categ='AC2')then

select ac2\_total\_seats,ac2\_booked\_seats into exist,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice'No seats left';

return ac2\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set ac2\_booked\_seats=ac2\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return ac2\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days);

end if;

elsif(categ='AC3')then

select ac3\_total\_seats,ac3\_booked\_seats into exist,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice 'No seats left';

return ac3\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set ac3\_booked\_seats=ac3\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return ac3\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days);

end if;

elsif(categ='CC')then

select cc\_total\_seats,cc\_booked\_seats into exist ,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice'No seats left';

return cc\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set cc\_booked\_seats=cc\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return cc\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days);

end if;

elsif(categ='EC')then

select ec\_total\_seats,ec\_booked\_seats into exist ,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice 'No seats left';

return ec\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set ec\_booked\_seats=ec\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return ec\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days);

end if;

elsif(categ='SL')then

select sl\_total\_seats,sl\_booked\_seats into exist ,left from seat\_class2 where(train\_id = t\_id and working\_day=days);

if(exist = left)then

raise notice 'No seats left';

return sl\_price from seat\_class2 where(train\_id = t\_id and working\_day=days) ;

else

update seat\_class2 set sl\_booked\_seats=sl\_booked\_seats+1

where(train\_id = t\_id and working\_day=days);

return sl\_price+100000 from seat\_class2 where(train\_id = t\_id and working\_day=days);

end if;

else

raise notice 'No such option';

return 0;

end if;

end;

$book\_seat$

language plpgsql

--How to call?

select book\_seat(10000,1,'AC1')

**Function - 3 :** Function to avoid duplicacy of user\_id (validating function)

Code:

create or replace function check\_user\_id(u\_id int)

returns int as $check\_user\_id$

declare

is\_pre int;

begin

is\_pre :=0;

select count(\*) into is\_pre from user1 where user\_id=u\_id;

if(is\_pre=1) then

raise notice 'Already exists';

else

raise notice 'New user';

end if;

return is\_pre;

end;

$check\_user\_id$ language plpgsql;

-- How to call?

select check\_user\_id(1000);

**Function -4:** Function to calculate total revenue for selected days generated to the railways per day.

Code:

-----------------------total revenue generated for a particular train on a day--------------------------------------

create or replace function revenue(t\_id int, dofweek int) returns int

as $revenue$

begin

return (seat\_class2.ac1\_price\*seat\_class2.ac1\_booked\_seats) +

(seat\_class2.ac2\_price\*seat\_class2.ac2\_booked\_seats) +

(seat\_class2.ac3\_price\*seat\_class2.ac3\_booked\_seats) +

(seat\_class2.cc\_price\*seat\_class2.cc\_booked\_seats) +

(seat\_class2.ec\_price\*seat\_class2.ec\_booked\_seats) +

(seat\_class2.sl\_price\*seat\_class2.sl\_booked\_seats)

from seat\_class2 where train\_id=t\_id and working\_day = dofweek;

end;

$revenue$

language plpgsql

select revenue(10001,2)

**Function - 5** : Function to display all the trains between 2 given dates with validations.

Code:

-----------------------------------train running between two dates---------------------------------------------------------------------

create or replace function train\_btwn\_date(f\_date date, t\_date date)

returns table(

t\_id int,

t\_name varchar(20),

s\_station int,

e\_station int,

day\_int int

)

as $train\_btwn\_date$

declare

day1 int;

day2 int;

begin

select extract(dow from f\_date) into day1;

select extract(dow from t\_date) into day2;

if(day1<day2) then

return query select distinct train1.train\_id,train1.train\_name,train1.starting\_station\_id,

train1.ending\_station\_id,seat\_class2.working\_day from train1,seat\_class2

where seat\_class2.working\_day >= day1 and seat\_class2.working\_day <= day2;

else

return query select distinct train1.train\_id,train1.train\_name,train1.starting\_station\_id,

train1.ending\_station\_id,seat\_class2.working\_day from train1,seat\_class2

where seat\_class2.working\_day >= day2 and seat\_class2.working\_day <= 7 and

seat\_class2.working\_day >= 0 and seat\_class2.working\_day <= day1;

end if;

end;

$train\_btwn\_date$

language plpgsql

select train\_btwn\_date('11-04-2020','15-04-2020')

**Function - 6:** Trains and its details running between given stations.

Code:

------------------------------------trains running between two stations----------------------------------------------

create or replace function train\_btwn(stat1 varchar, stat2 varchar)

returns table(

t\_id int,

t\_name varchar(20),

on\_mon char(1),

on\_tue char(1),

on\_wed char(1),

on\_thr char(1),

on\_fri char(1),

on\_sat char(1),

on\_sun char(1)

) as $train\_btwn$

begin

return query select train\_id,train\_name,mon,tue,wed,thu,fri,sat,sun from train1

where(starting\_station\_id = (select station\_id from station1 where station\_name=stat1)

and ending\_station\_id = (select station\_id from station1 where station\_name=stat2));

end;

$train\_btwn$

language plpgsql

select train\_btwn('mumbai','delhi')

**Function - 7** : Function to cancel tickets for users. Also this function sets the very next person in the waiting list to confirmed status.  
Code:

create or replace function cancel\_ticket(ticket int)

returns int as

$cancel\_ticket$

declare

exist int;

categ varchar(3);

sta char;

begin

sta = status from passenger1 where pnr = ticket;

if(sta = 'W') then

delete from passenger1 where pnr=ticket;

return 0;

end if;

categ = seat\_category from passenger1 where pnr = ticket;

exist = min(pnr) from passenger1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

dayno=(select dayno from passenger1 where pnr=ticket) and

seat\_category=(select seat\_category from passenger1 where pnr=ticket) and

status='W';

if(exist is NULL) then

if(categ='AC1') then

update seat\_class2 set ac1\_booked\_seats = ac1\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

elsif(categ='AC2') then

update seat\_class2 set ac2\_booked\_seats = ac2\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

elsif(categ='AC3') then

update seat\_class2 set ac3\_booked\_seats = ac3\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

elsif(categ='CC') then

update seat\_class2 set cc\_booked\_seats = cc\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

elsif(categ='EC') then

update seat\_class2 set ec\_booked\_seats = ec\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

elsif(categ='SL') then

update seat\_class2 set sl\_booked\_seats = sl\_booked\_seats-1 where

train\_id=(select train\_id from passenger1 where pnr=ticket) and

working\_day=(select dayno from passenger1 where pnr=ticket);

end if;

delete from passenger1 where pnr=ticket;

return 0;

else

update passenger1 set status='C' where pnr=exist;

end if;

delete from passenger1 where pnr=ticket;

return 0;

end;

$cancel\_ticket$

language plpgsql;

**Function - 8 :** Function to obtain/inquire about booked tickets. Print Ticket.

Code:

------------------function for showing booked tickets-------------------

create or replace function show\_tickets(u\_id int) returns

table(

pnr\_no int,

us\_id int,

u\_name varchar(20),

d\_of\_w int,

tr\_id int,

tr\_name varchar(20),

f\_stat varchar(20),

t\_stat varchar(20),

categ varchar(3),

stat char(1),

price int

) as $show\_ticket$

declare

flag int;

begin

return query select distinct passenger1.pnr,user1.user\_id,user1.name,passenger1.dayno,passenger1.train\_id,

train1.train\_name,aa.station\_name,bb.station\_name,passenger1.seat\_category,

passenger1.status,passenger1.price

from station1 bb inner join

station1 aa inner join

train1 inner join

passenger1 inner join user1

on passenger1.passenger\_id=user1.user\_id

on train1.train\_id=passenger1.train\_id

on aa.station\_id = train1.starting\_station\_id

on bb.station\_id = train1.ending\_station\_id

where u\_id=user\_id and u\_id=passenger\_id;

end;

$show\_ticket$

language plpgsql

select show\_tickets(1002)

**Function - 9 :** Function to obtain reservation chart details for any given train.

Code:

create or replace function all\_pass\_of\_train(t\_id int, days int)

returns table(

pass\_id int,

pass\_name varchar(20),

categ varchar(3)

)

as $all\_pass\_of\_trian$

begin

return query select

passenger1.passenger\_id,user1.name,passenger1.seat\_category

from passenger1 inner join user1 on passenger1.passenger\_id = user1.user\_id

where passenger1.train\_id = t\_id and passenger1.dayno=days

order by passenger1.seat\_category;

end;

$all\_pass\_of\_trian$

language plpgsql

select \* from all\_pass\_of\_train(10001,2)

**Function - 10 :** AS discussed this is our special function. If a user wants to go Station A to C but in between take a via route of another station B and with some time to halt at this station. So his new route is from station A to B to C. This function gives the output of such trains.

Code:

create or replace function timechange(t\_1 float)

returns float

as $timechange$

declare

now\_time numeric(4,2);

begin

now\_time = t\_1-trunc(t\_1);

return trunc(now\_time/0.6) + trunc(t\_1) + MOD(now\_time,0.6);

end;

$timechange$

language plpgsql

insert into train1 values(10007,'Check1',101,102,10.5,1.2,'Y','N','Y','N','Y','N','Y');

insert into train1 values(10008,'Check2',101,102,12.2,1.4,'Y','N','Y','N','Y','N','Y');

insert into seat\_class2 values(10007,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

insert into seat\_class2 values(10008,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

create or replace function dis\_two\_train(st1\_id int,st2\_id int,st3\_id int,dayno int)

returns table(

tra1\_id int,

tra1\_name varchar(20),

tra1\_sta float,

tra1\_stop float,

tra2\_id int,

tra2\_name varchar(20),

tra2\_sta float,

tra2\_stop float

)

as $dis\_two\_train$

Begin

return query select aa.train\_id, aa.train\_name, aa.starting\_time,

timechange(aa.starting\_time + aa.journey\_time),

bb.train\_id, bb.train\_name, bb.starting\_time,

timechange(bb.starting\_time + bb.journey\_time)

from seat\_class2 inner join

train1 aa inner join train1 bb

on aa.mon = bb.mon

on seat\_class2.train\_id = aa.train\_id

where aa.mon='Y' and bb.mon='Y' and seat\_class2.working\_day=dayno and

bb.starting\_time >= timechange(aa.starting\_time+aa.journey\_time)

and aa.starting\_station\_id=st1\_id and aa.ending\_station\_id=st2\_id

and bb.starting\_station\_id=st2\_id and bb.ending\_station\_id=st3\_id;

end;

$dis\_two\_train$

language plpgsql

select dis\_two\_train(101,102,103,1)

**TRIGGERS:**

**Trigger -1:** Trigger to validate all user inputs.

Code:

$check\_user1\_proc$

begin

if(new.user\_id <1000 or new.user\_id>9999) then

raise exception 'User ID invalid';

end if;

if(new.name is NULL) then

raise exception 'Name not written';

end if;

if(new.gender <> 'F' and new.gender <> 'M') then

raise exception 'No such Gender ';

end if;

if(new.mobile\_no >999999 or new.mobile\_no<100000) then

raise exception 'Mobile number invalid';

end if;

return new;

end;

$check\_user1\_proc$

language plpgsql;

create trigger check\_user1

before insert or update on user1

for each row execute procedure check\_user1\_proc();

**Trigger - 2 :** Trigger to validate all train details using functions + triggers (handled by administrator) .

Code:

create or replace function check\_train1\_proc() returns trigger as

$train\_user1\_proc$

begin

if(new.train\_name is NULL) then

raise exception 'Train name not written';

end if;

if(new.starting\_station\_id = new.ending\_station\_id) then

raise exception 'Train is having same starting station and ending station';

end if;

if(new.starting\_time > 24 or new.starting\_time < 0) then

raise exception 'Starting time is invalid';

end if;

if(new.mon='N' and new.tue='N' and new.wed='N' and new.thu='N' and new.fri='N' and new.sat='N' and new.sun='N')then

raise exception 'Train is not working on any day';

end if;

return new;

end;

$train\_user1\_proc$

language plpgsql;

create trigger check\_train1 before insert or update on train1

for each row execute procedure check\_train1\_proc();

**Trigger - 3 :** Trigger to delete any user for deleting account by admin or user.

This trigger calls cancel tickets option so the very next person in the waiting list gets his status confirmed.

Code:

create or replace function delete\_user() returns trigger as

$delete\_user$

declare

Exist int;

c\_delete\_user cursor for select pnr from passenger1 where passenger\_id=old.user\_id;

rec\_user passenger1.pnr%type;

begin

open c\_delete\_user;

loop

fetch c\_delete\_user into rec\_user;

exit when not found;

Exist = cancel\_ticket(rec\_user);

delete from passenger1 where pnr=rec\_user;

end loop;

close c\_delete\_user;

return old;

end;

$delete\_user$

language plpgsql;

create trigger delete\_user before delete on user1

for each row execute procedure delete\_user();

delete from user1 where user\_id=1808

**Trigger -4** : Trigger for maintaining track of all the deleted/canceled bookings or reservations made.

Code:

----------------------------------------record for showing list of deleted tickets--------------------------------

create or replace function del\_tic\_records() returns trigger as

$del\_tic\_records$

declare

fname varchar(20);

begin

select name into fname from user1 where user\_id=old.passenger\_id;

insert into deleted\_tickes values(old.pnr, old.passenger\_id,fname,old.train\_id,old.dayno,

old.seat\_category,old.status,localtimestamp);

return old;

end;

$del\_tic\_records$

language plpgsql;

create trigger delete\_t before delete on passenger1

for each row execute procedure del\_tic\_records();

select \* from deleted\_tickes

**Trigger -5** : Trigger for checking if the user wallet balance is at least that of ticket price so that he can book this ticket (balance >= fare).

create or replace function check\_user1\_bal() returns trigger as

$check\_user1\_proc$

begin

if(TG\_OP = 'INSERT') then

if(new.wallet\_bal < 0) then

raise exception 'Invalid balance';

end if;

elsif(TG\_OP = 'UPDATE') then

if(new.wallet\_bal < 0) then

raise exception 'Insufficient balance';

end if;

end if;

return new;

end;

$check\_user1\_proc$

language plpgsql;

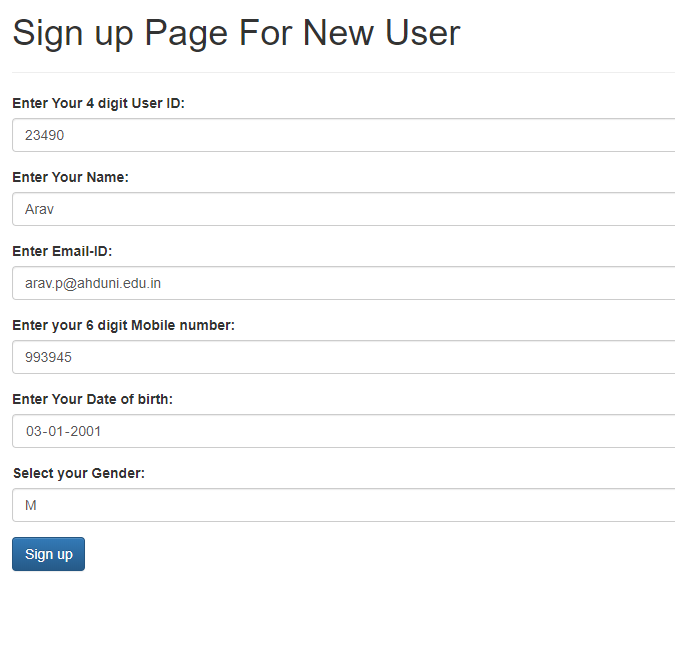
create trigger check\_balance

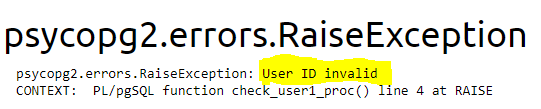
before insert or update on user1

for each row execute procedure check\_user1\_bal();

# ScreenShots of error Generated on FrontEnd due to Triggers

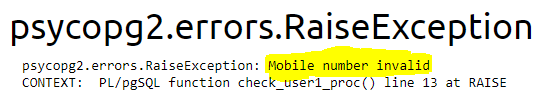
1. Due to invalid input from the user side , we have created a trigger which checks user inputs details for validation.

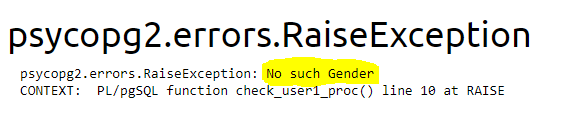




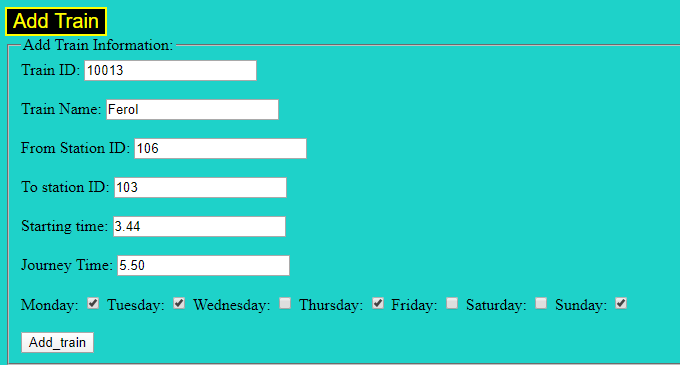
Here user id should be 4 digit but on entering 5 digit user id an error message is generated.

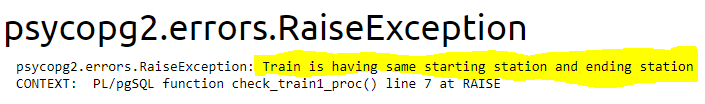
Similarly other displayed messages are:

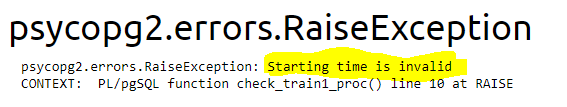


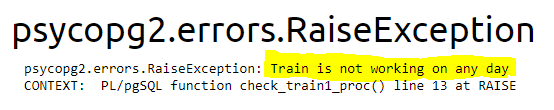


Invalid Inputs for train

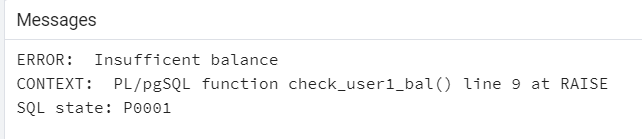






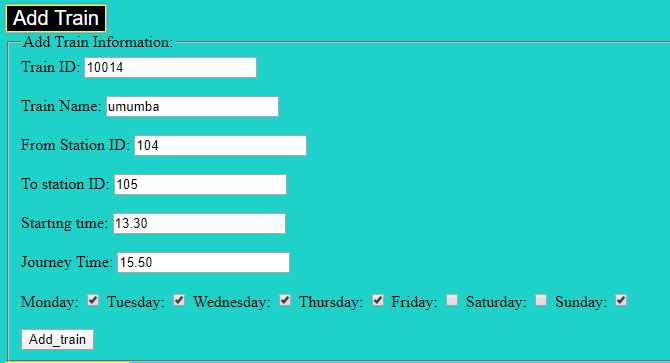


Appropriate balance does not exist then:



# Functions/Procedures Screenshots of frontend.

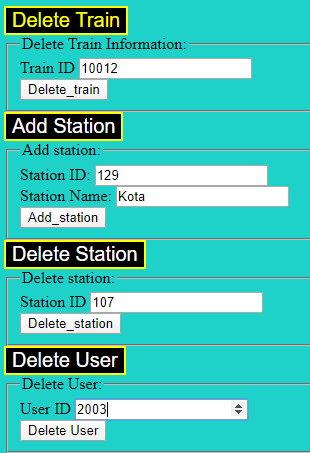
1. **Add a train**



1. **Adding seat category to that train**

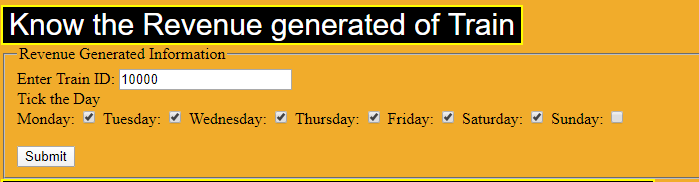


1. **Similarly other data manipulation of trains for admin:**

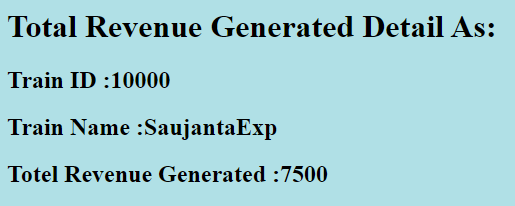


1. **Revenue Generated to the Railways (admin usage)**

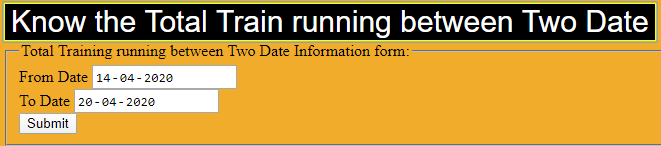
Say the Admin wants to calculate the revenue generated by a train till now. Then he has to provide train id and select day for including revenue calculation.



So for this train we get output as

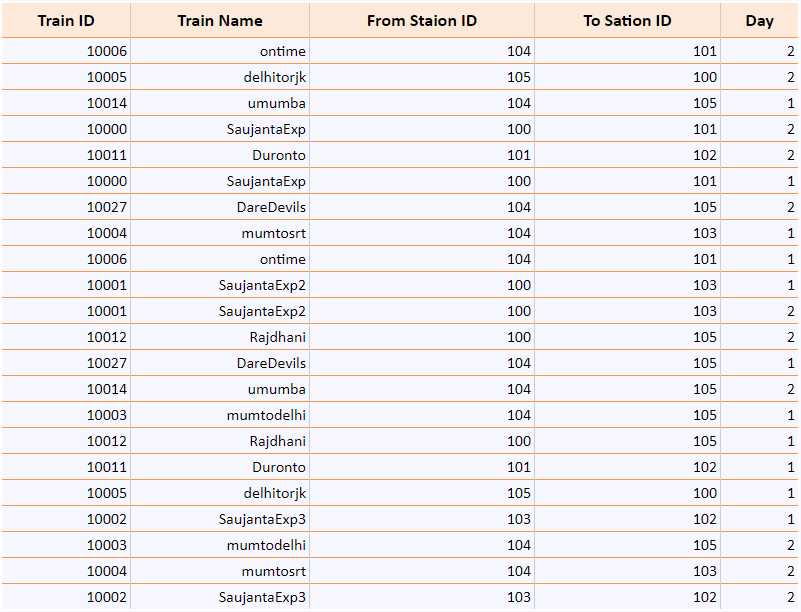


1. **Functionality for getting the number of trains and its details between 2 dates:**

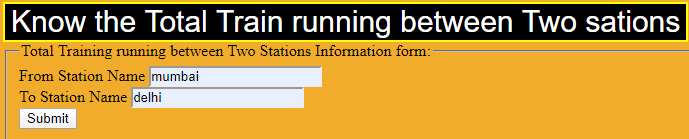


So the output generated is :

Here 1 stands for monday, 2 stands for tuesday..

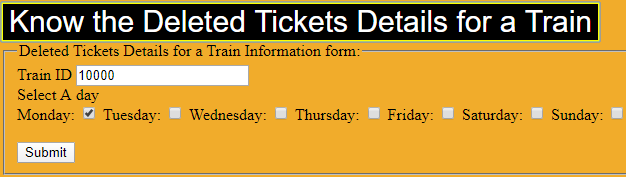


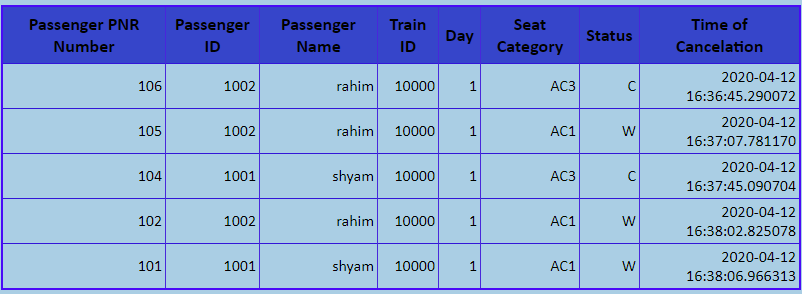
1. **Input two station name to know total trains and its details running between them**





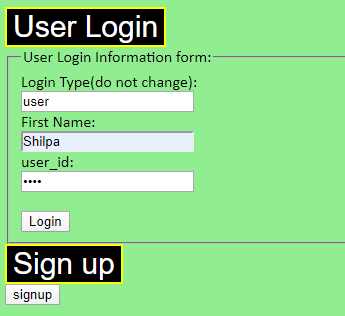
1. **Input a train id and select a day to see the details of passengers who have deleted their tickets.** Details includes name,date,seat category, time



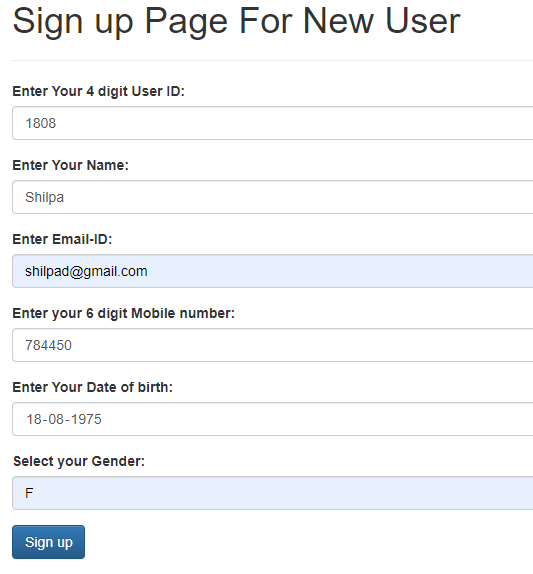


1. **Login Module**

User can create his new account or login into an existing account.

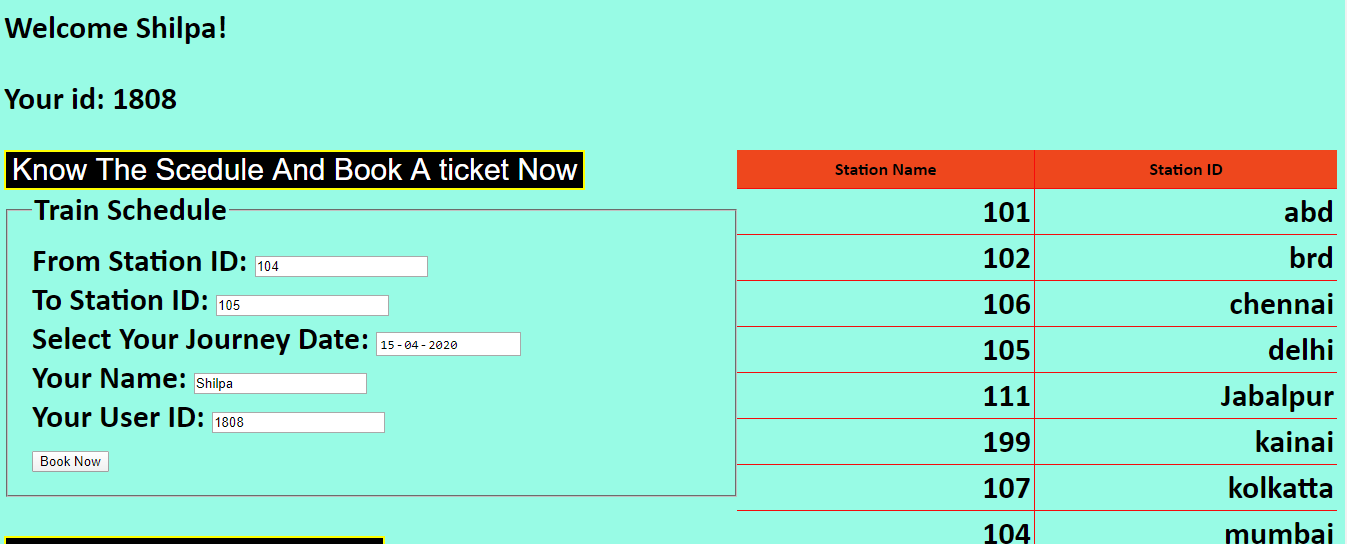


Sign up page:

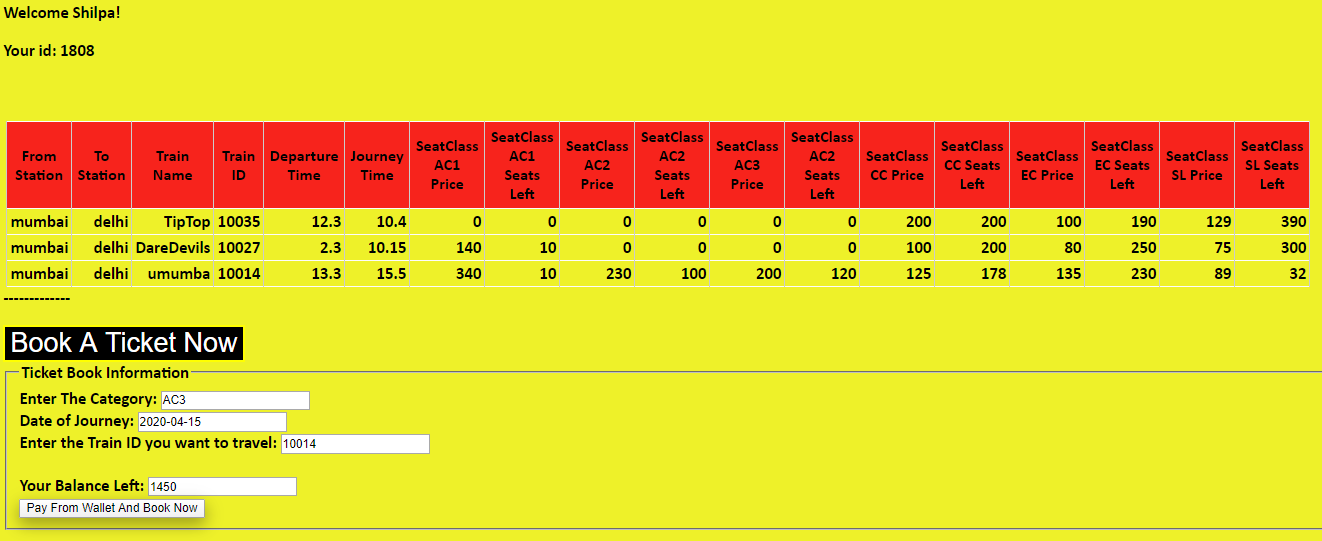


1. **Schedule For user to Book Ticket**

* Input from and to station and date to know the schedule of trains and book a train.
* Fare is deducted from his wallet (if he has appropriate balance or error).

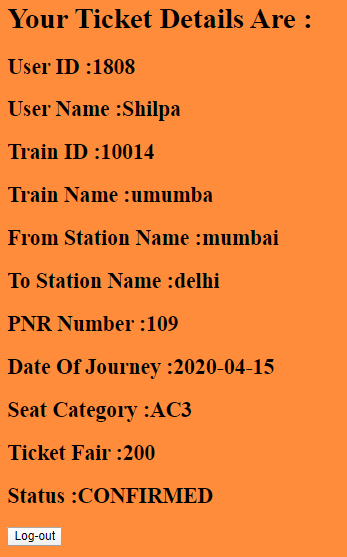


Output is generated as:



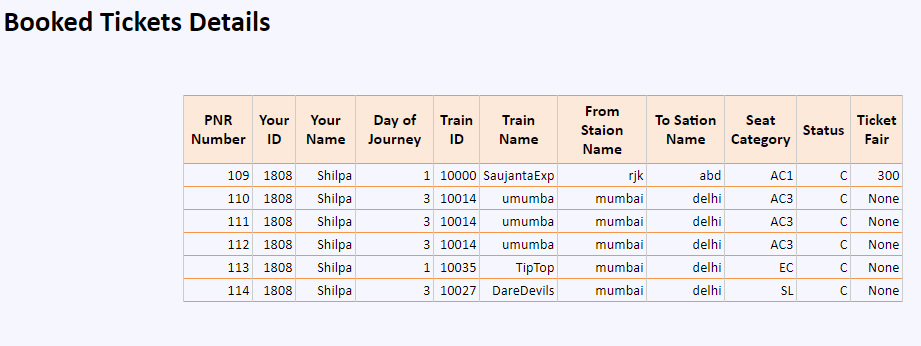
1. **Generating Ticket**

Once user books his ticket then his e-ticket is generated as:



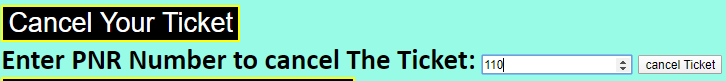
1. **Booking History with reservation status**

This function is about showing all the tickets booked by the user till now with every details including status.

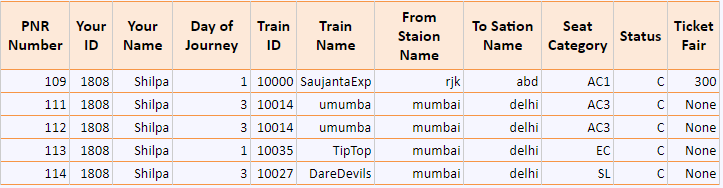


1. **Cancelation of Booking**

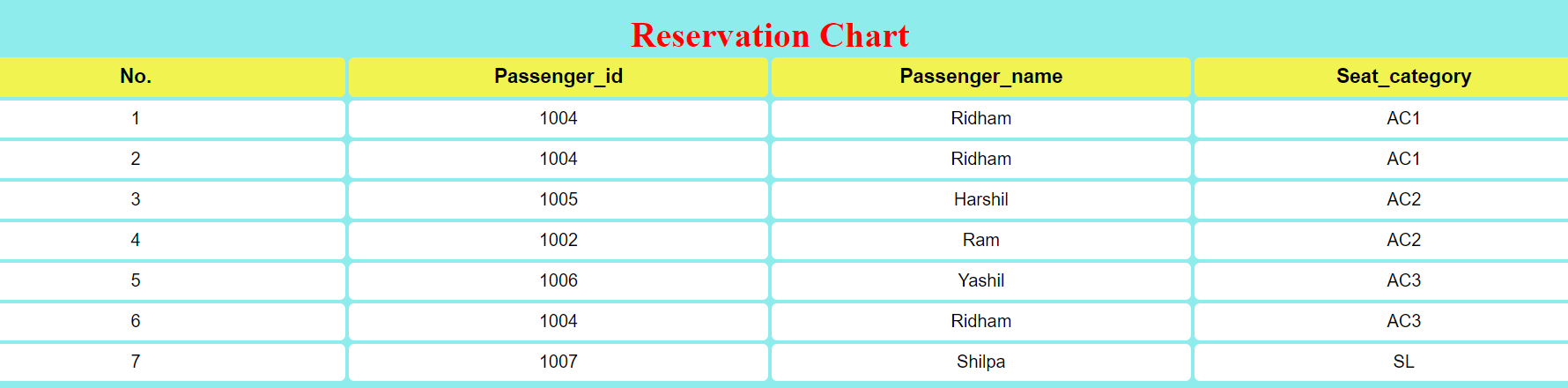
Cancel ticket.



So we can see that this ticket is deleted



1. **Display Reservation Chart**



This is the reservation chart which is used to show all the bookings made in any train on a given day.

1. **Special 3 station - (used for connecting trains from A -> B and B -> C when user wants halt at station B):**

As discussed, users can input 3 stations with some waiting/working time in station 2. So this function gives the output as the first train from station 1 to 2 with journey time. Users can have some personal work done at station 2. Then another train from station 2 to station 3 has starting time after that user has done his work.

