

IIT Railway System

IITRS

Amish Ashish Saxena, Rudr Tiwari, Padala SSSS V Sri Vishnu Subhash

Roll no : 111901008 , 111901044, 111901038

Indian Institute of Technology Palakkad, Kerala, India

13th March 2022

Overview

Here we briefly describe about the features which we have implemented in our database system. Our reservation system will allow the user to make an account so that all the tickets he/she books can be viewed at once. A user has to be registered in order to book tickets. An unregistered user still check the availability of the trains. The model of our train follows that of a real train in India. A train has multiple classes and there are multiple coaches of each class type in a train. Since a train moves between several railway stations, we give the flexibility of choosing the start and end station to the user.

Implementation for running the application

We run our entire system using python. Thus all the update, insert and delete queries are called by python scripts. This is done using sqlalchemy package. We also implement a simple terminal based gui (tui) to implement the working of our database system. This is done using the picotui package. Our final system consisting of the entire database along with the gui is shown in the end.

ER Diagram

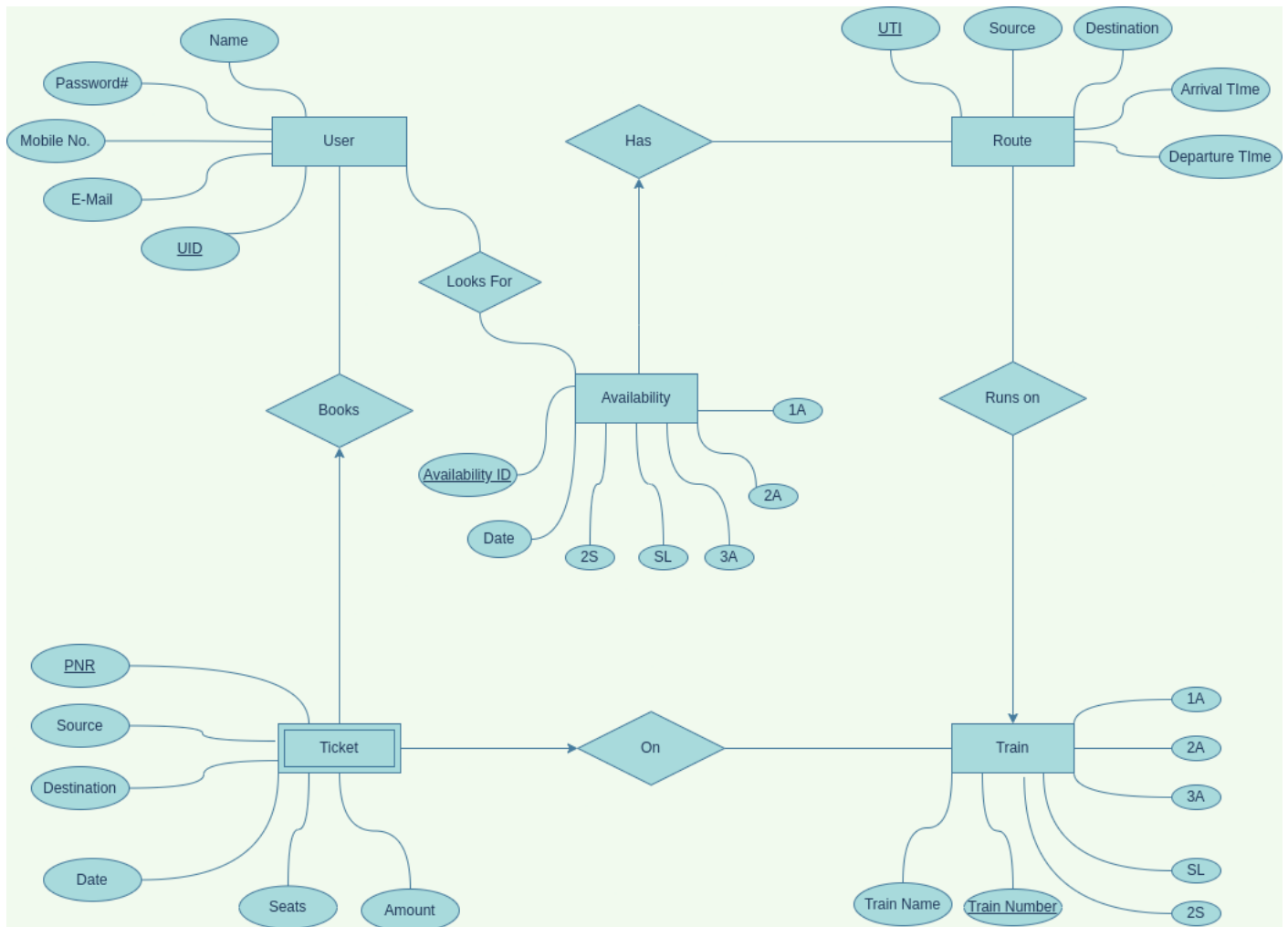


Figure 1: ER Diagram

1. Ticket is a weak entity since it is dependent on both the user as well as the train.
2. Since, a user can book multiple number of tickets, the relation between the User and Ticket is one-to-many.
3. Similarly, ticket-to-train , route-to-train and availability-to-route are many-to-one relations.

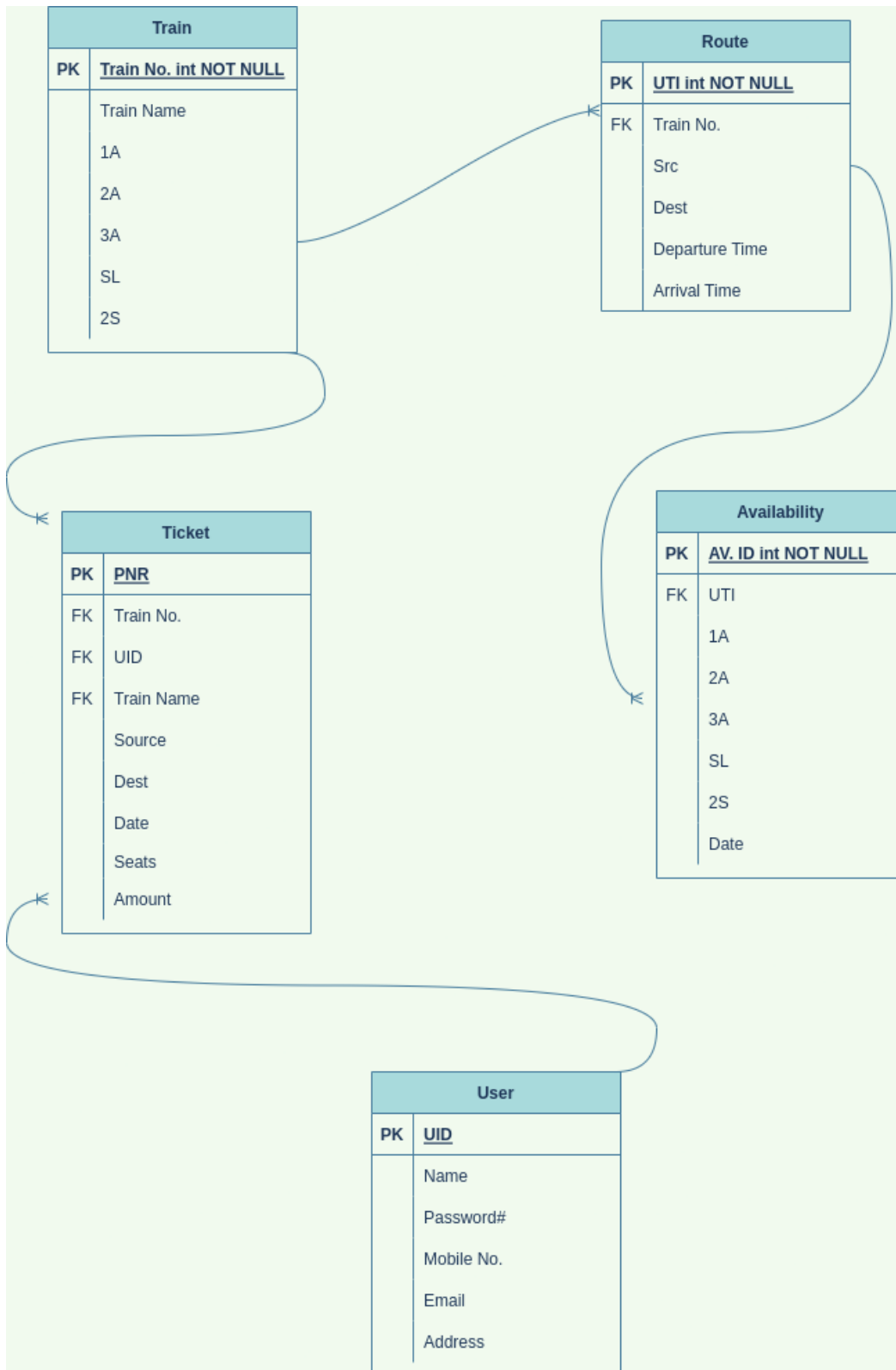


Figure 2: Table Diagram

Table Schema

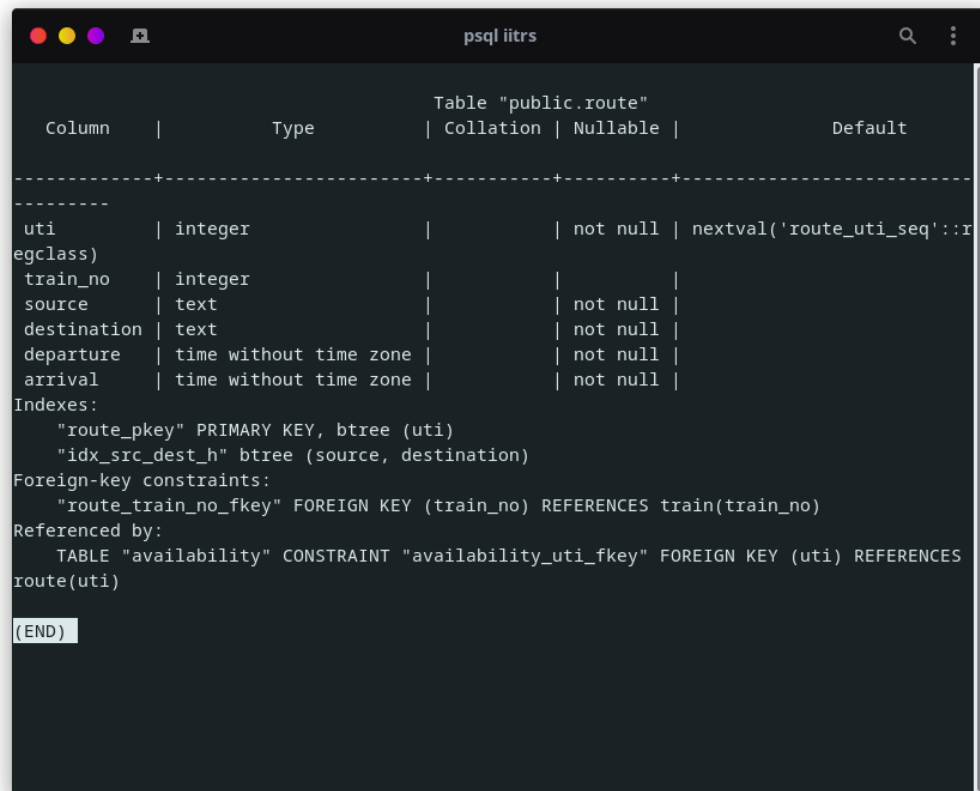
The Train Table

```
psql iitrs
iitrs=# \d train
Table "public.train"
  Column   | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
train_no  | integer |           | not null |
train_name | text    |           | not null |
first_ac  | integer |           |          |
second_ac | integer |           |          |
third_ac  | integer |           |          |
sleeper   | integer |           |          |
general   | integer |           |          |
Indexes:
    "train_pkey" PRIMARY KEY, btree (train_no)
Check constraints:
    "train_first_ac_check" CHECK (first_ac >= 0)
    "train_general_check" CHECK (general >= 0)
    "train_second_ac_check" CHECK (second_ac >= 0)
    "train_sleeper_check" CHECK (sleeper >= 0)
    "train_third_ac_check" CHECK (third_ac >= 0)
    "train_train_no_check" CHECK (train_no > 9999)
Referenced by:
    TABLE "route" CONSTRAINT "route_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)
    TABLE "ticket" CONSTRAINT "ticket_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)
    TABLE "train_journey" CONSTRAINT "train_journey_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)
iitrs=#
```

Figure 3: Train Table Schema

The train table stores the basic data corresponding to all the trains. It has train number column as it's primary key and a column for train's name. The train table also stores the number of coaches for each of the classes (First AC, Second AC, Third AC, Sleeper and General) for that train.

The Route Table



Column	Type	Collation	Nullable	Default
uti	integer		not null	nextval('route_uti_seq'::regclass)
train_no	integer			
source	text		not null	
destination	text		not null	
departure	time without time zone		not null	
arrival	time without time zone		not null	

Indexes:

- "route_pkey" PRIMARY KEY, btree (uti)
- "idx_src_dest_h" btree (source, destination)

Foreign-key constraints:

- "route_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)

Referenced by:

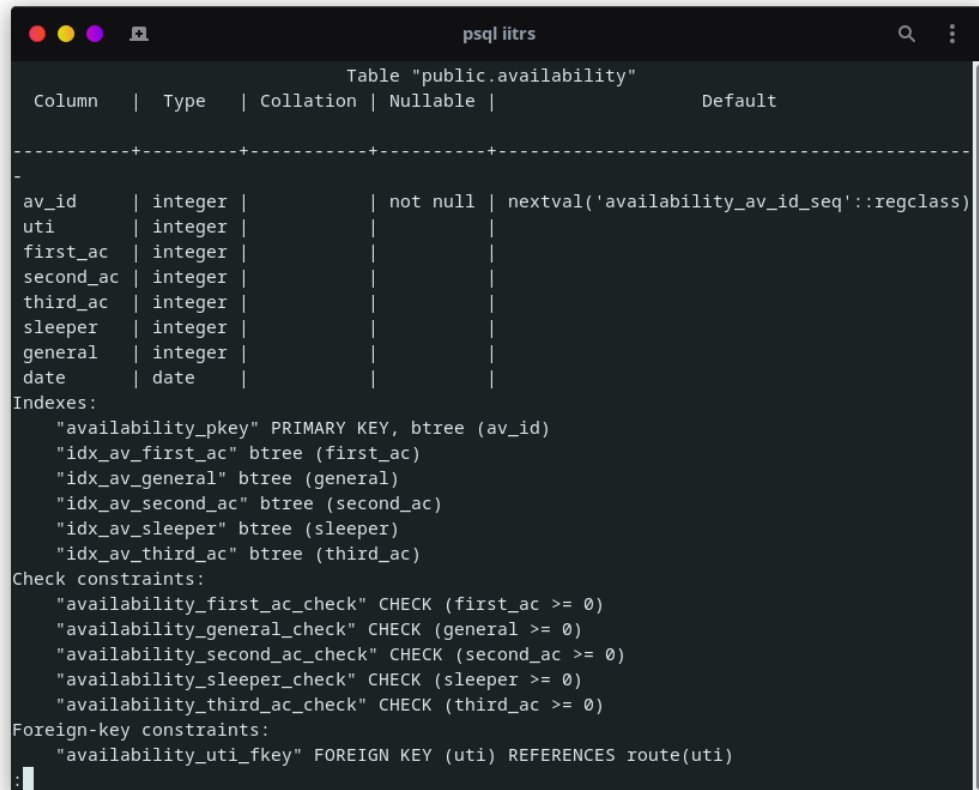
- TABLE "availability" CONSTRAINT "availability_uti_fkey" FOREIGN KEY (uti) REFERENCES route(uti)

(END)

Figure 4: Route Table Schema

The route table is used to store the journey of all the trains in it. The route table has UTI (Unique Train ID) as it's primary key. It will also have a train number column which references the from the train table as foreign key. The table will have multiple instances of a single train including all the stations it visits. Thus if a train has the route $A \rightarrow B \rightarrow C$ then it will include entries for $A \rightarrow B$, $A \rightarrow C$ and $B \rightarrow C$ for that train number, each of them having a unique UTI number. The table also includes the departure and arrival times (without time zone) for the source source station.

The Availability Table



Column	Type	Collation	Nullable	Default
av_id	integer		not null	nextval('availability_av_id_seq'::regclass)
uti	integer			
first_ac	integer			
second_ac	integer			
third_ac	integer			
sleeper	integer			
general	integer			
date	date			

Indexes:

- "availability_pkey" PRIMARY KEY, btree (av_id)
- "idx_av_first_ac" btree (first_ac)
- "idx_av_general" btree (general)
- "idx_av_second_ac" btree (second_ac)
- "idx_av_sleeper" btree (sleeper)
- "idx_av_third_ac" btree (third_ac)

Check constraints:

- "availability_first_ac_check" CHECK (first_ac >= 0)
- "availability_general_check" CHECK (general >= 0)
- "availability_second_ac_check" CHECK (second_ac >= 0)
- "availability_sleeper_check" CHECK (sleeper >= 0)
- "availability_third_ac_check" CHECK (third_ac >= 0)

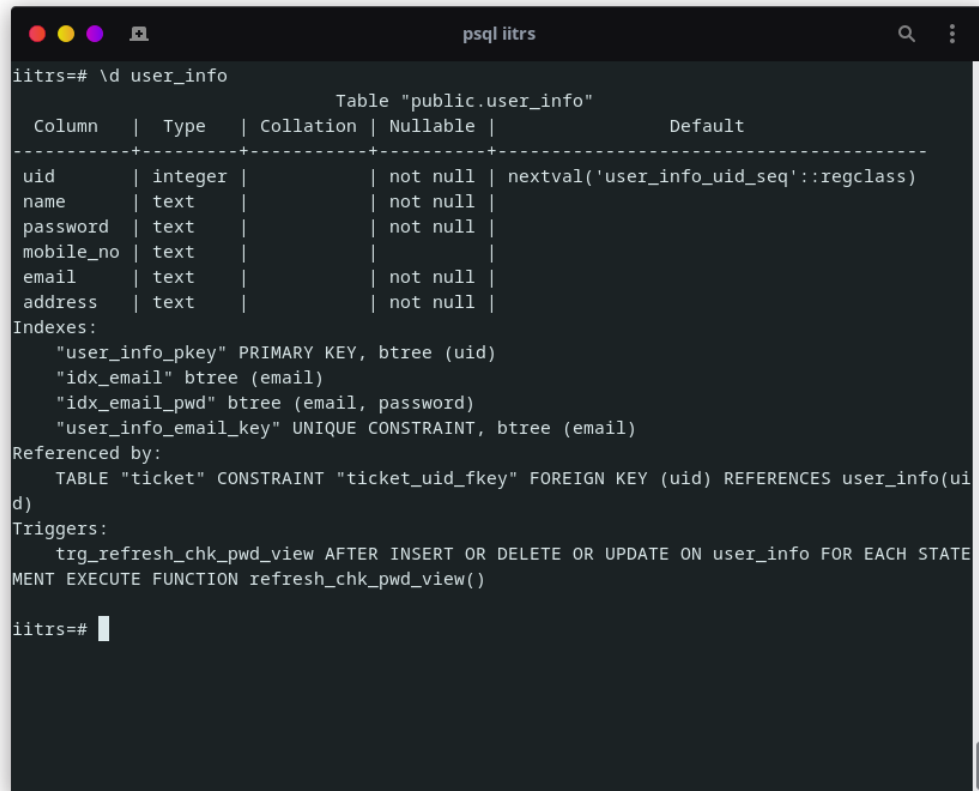
Foreign-key constraints:

- "availability_uti_fkey" FOREIGN KEY (uti) REFERENCES route(uti)

Figure 5: Availability Table Schema

The availability table is basically for storing the number of seats available in a train running on a particular date. To implement this, the table references the UTI number from the route table along with the dates on which it runs in an year stored in a separate column. It has av_id (availability id) as it's primary key, which is a unique id for each (uti, date) pair. The availability is stored independently for each of the coaches.

The User Table



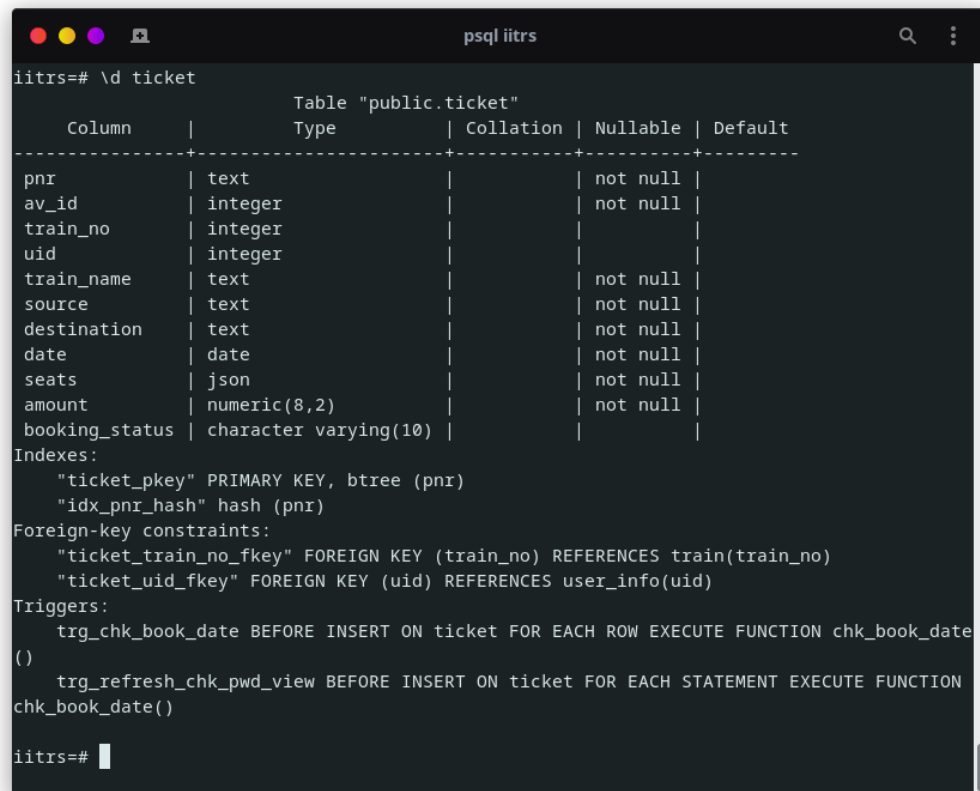
```
psql iitrs
iitrs=# \d user_info
                                Table "public.user_info"
  Column  | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
 uid      | integer |           | not null | nextval('user_info_uid_seq'::regclass)
 name     | text    |           | not null |
 password | text    |           | not null |
 mobile_no | text    |           |          |
 email    | text    |           | not null |
 address  | text    |           | not null |
Indexes:
    "user_info_pkey" PRIMARY KEY, btree (uid)
    "idx_email" btree (email)
    "idx_email_pwd" btree (email, password)
    "user_info_email_key" UNIQUE CONSTRAINT, btree (email)
Referenced by:
    TABLE "ticket" CONSTRAINT "ticket_uid_fkey" FOREIGN KEY (uid) REFERENCES user_info(uid)
Triggers:
    trg_refresh_chk_pwd_view AFTER INSERT OR DELETE OR UPDATE ON user_info FOR EACH STATEMENT EXECUTE FUNCTION refresh_chk_pwd_view()

iitrs=#
```

Figure 6: User Table Schema

The user table stores the data entered by a user at the time of registering in the portal and is used to implement the login functionality. Each user has a unique id (UID) as it's primary key. Other fields which are included in the user_info are the user's name, his / her password (in encrypted form), mobile number (optional), email (unique) and address.

The Ticket Table



```
iitrs=# \d ticket
```

Column	Type	Collation	Nullable	Default
pnr	text		not null	
av_id	integer		not null	
train_no	integer			
uid	integer			
train_name	text		not null	
source	text		not null	
destination	text		not null	
date	date		not null	
seats	json		not null	
amount	numeric(8,2)		not null	
booking_status	character varying(10)			

Indexes:

- "ticket_pkey" PRIMARY KEY, btree (pnr)
- "idx_pnr_hash" hash (pnr)

Foreign-key constraints:

- "ticket_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)
- "ticket_uid_fkey" FOREIGN KEY (uid) REFERENCES user_info(uid)

Triggers:

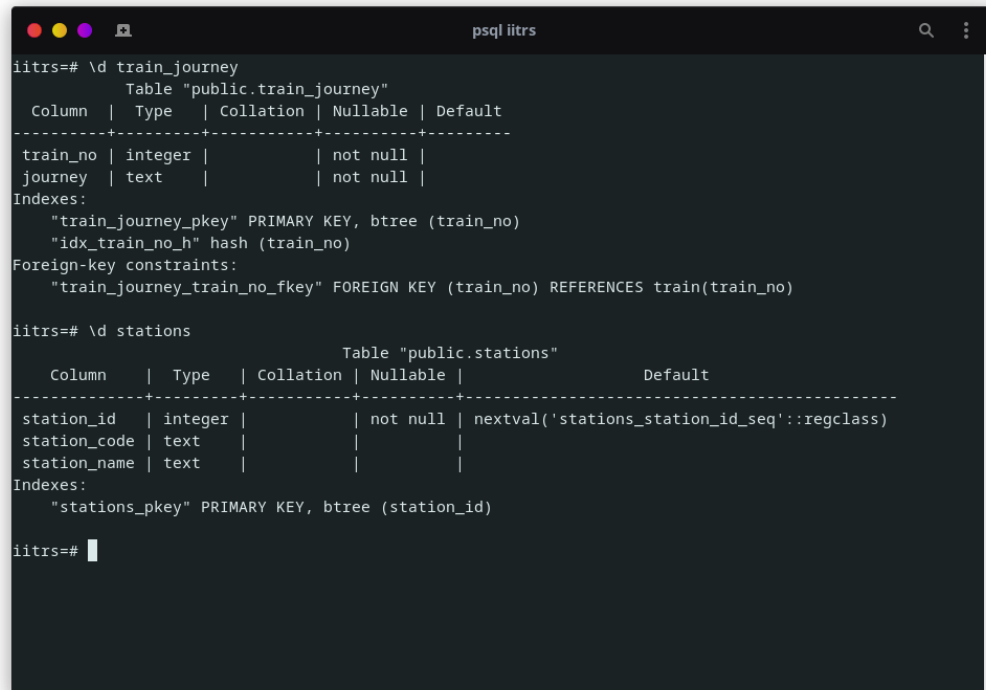
- trg_chk_book_date BEFORE INSERT ON ticket FOR EACH ROW EXECUTE FUNCTION chk_book_date()
- trg_refresh_chk_pwd_view BEFORE INSERT ON ticket FOR EACH STATEMENT EXECUTE FUNCTION chk_book_date()

```
iitrs=#
```

Figure 7: Ticket Table Schema

The ticket table is the pool to store the tickets booked by all the registered user in the database. As a result, the table should obviously inherit the train number and UID from the user and the train table as it's foreign key. The other basic information like train number, source, destination, date and the amount in separate columns. Other information about the travellers (like age, name, gender, seat number) will be stored in the seats column as a JSON object. It should be noted that one ticket can include multiple passengers.

Helper Tables



```
psql iitrs
iitrs=# \d train_journey
          Table "public.train_journey"
  Column | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
 train_no | integer |           | not null |
 journey  | text    |           | not null |
Indexes:
    "train_journey_pkey" PRIMARY KEY, btree (train_no)
    "idx_train_no_h" hash (train_no)
Foreign-key constraints:
    "train_journey_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no)

iitrs=# \d stations
          Table "public.stations"
  Column | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
 station_id | integer |           | not null | nextval('stations_station_id_seq'::regclass)
 station_code | text    |           |          |
 station_name | text    |           |          |
Indexes:
    "stations_pkey" PRIMARY KEY, btree (station_id)

iitrs=#
```

Figure 8: Helper Table Schema

We also have two helper tables which were added by us later in order to maintain some extra functionality. There are the stations table and the journey table. The journey table stores the entire journey of a train using a string. This is used while booking and cancellation since we need to update the availability of many routes for a train once it is booked. The stations helps maintaining a database consisting of all the stations.

Functionalities

View Availability

A user does not need to be registered to view the availability of different trains and seats across a route.

Available trains between two stations

By simply providing the corresponding station for the Source and the Destination, the user can see the list of available trains between two stations.

Availability of seats

The system takes the input containing :

1. Journey Origin Station
2. Journey Termination Station
3. Date of Journey

This returns a list of all the different trains that have available seats corresponding to the requested journey by the user. The user can then select a travel class to find the available number of seats related to it.

This request is handled by querying the join of the *route* and the *available* table.

The user can then login or register and proceed to book the ticket for a desired journey.

Register

A user can register himself in the system by using the Register feature. Upon entering the details specific to the user like E-mail, Name, Address, etc. along with a password, a user would be registered. The registered users' details are stored in the *user_info* table.

Login

In-order to book a ticket a user needs to be registered. A registered user can then login using the respective credentials to book the tickets.

The login can be handled by the *user* table where the information of the user is stored. A logged in user can access various features that are not available to a Guest User. Such as : Book Ticket, Cancel Ticket, View Booked Ticket History, etc.

Book Tickets

A user needs to be a registered user to book tickets. In order to book a ticket the user first needs to search for a train using `View Available seats` functionality and then select a particular travel class with sufficiently available number of seats as per their requirements.

The customer then has to select the number of seats required to book and then enter the details of all the passengers pertaining to each of the seats.

The book ticket functionality is atomic in nature and is handled using a transaction. This request is handled by a series of steps where first a function checks if the numbers of required seats by the user are available or not. Then the particular seats are allotted to the user and a corresponding entry is added to the tickets table. The allotted number of seats are then removed from the respective column in the availability table. If all goes well, then the transaction is completed and a ticket is successfully booked. Otherwise, an appropriate error is thrown prompting the user to try again, while the transaction is rolled back.

Booked Ticket History

A user can find all of the past bookings in the booked ticket history panel. This list of tickets will contain all the data of the tickets booked earlier. The user can access the list of booked tickets by applying filters like sorting according to date, train, etc. Even after a booked ticket is cancelled, the user can see the ticket and find the *Status* as Cancelled for the particular ticket.

These requests are handled by querying the ticket table.

Cancel Ticket

If a user wishes to cancel a ticket, he can do so by logging in with the same credentials that he used to book that ticket and then proceed to the cancellation option. The user will be required to enter the *PNR* number of the ticket he wishes to cancel. This operation is again atomic and is done using a transaction upon the *availability* and the *user_info* table.

After the ticket has been cancelled, the user can see the updated status of the ticket in the *Booked Ticket History* section as Cancelled.

Once, a ticket is cancelled, the corresponding seats are then added back to the availability so that they can be booked again.

User details

The user can view his own details once he is logged in. The details of the user apart from the password are displayed here.

This is done by querying the user_info table.

Change Password

The logged in user has the ability to change their password.

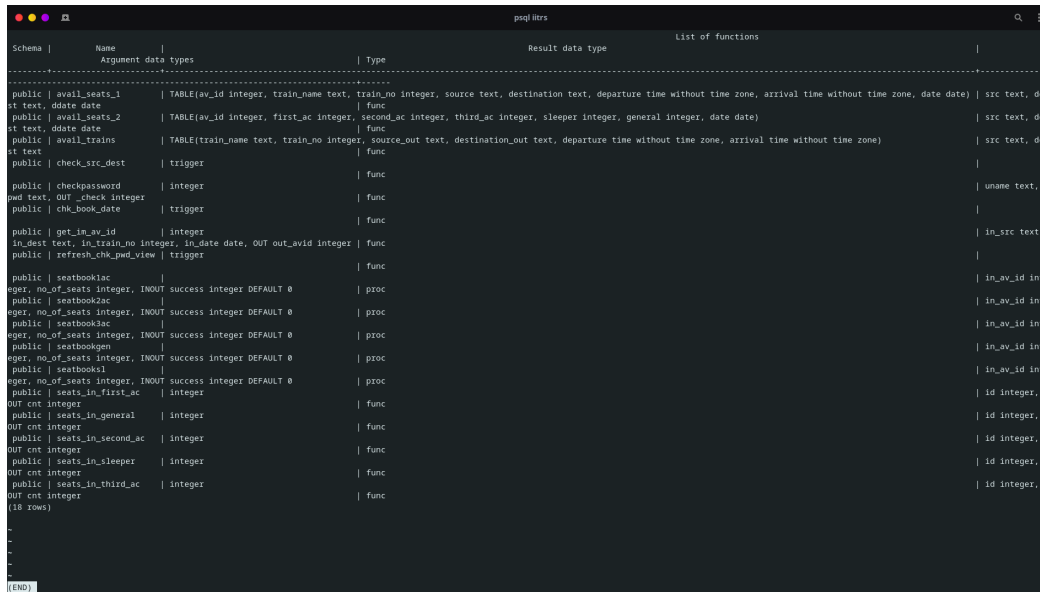
Roles

We define 3 level hierarchy of Roles for the database :

1. **Admin** : The admin has all the access to modify the entire database.
2. **Rail Manager** : The railway manager is responsible for updating the schedules for different trains (addition/updating). He does not possess the authority to modify the database schema.
3. **Registered User** : This user has all the permissions available to the un-registered user along with the authorization to book tickets. This helps in keeping track of the tickets each user has booked.
4. **Guest User** : This type of user can only view the available seats, search trains on different routes and lookup for already booked ticket's information using a PNR number.

Specifics of the Data-Base

Functions



Schema	Name	Argument data types	Type	Result data type	List of functions
public	avail_seats_1	TABLE(av_id integer, train_name text, train_no integer, source text, destination text, departure time without time zone, arrival time without time zone, date date)	func	src text, de	
public	avail_seats_2	TABLE(av_id integer, first_ac integer, second_ac integer, third_ac integer, sleeper integer, general integer, date date)	func	src text, de	
public	avail_trains	TABLE(train_name text, train_no integer, source_out text, destination_out text, departure time without time zone, arrival time without time zone)	func	src text, de	
public	check_src_dest		trigger		
public	checkpassword	integer	func		
public	chk_book_date	integer	func		
public	get_in_av_id	integer	func		
public	refresh_chk_pwd_view	integer	trigger		
public	seatbook1ac	integer	proc		
public	seatbook2ac	integer	proc		
public	seatbook3ac	integer	proc		
public	seatbook4ac	integer	proc		
public	seatbook5ac	integer	proc		
public	seats_in_general	integer	func		
public	seats_in_first_ac	integer	func		
public	seats_in_second_ac	integer	func		
public	seats_in_sleeper	integer	func		
public	seats_in_third_ac	integer	func		

Figure 9: Functions in the database

The following functions are used to implement the above mentioned functionalities:

- **avail trains** : This queries a view called as avail.trains_view and returns all trains between two stations.
Feature Implemented : *Available trains between two stations*
- **avail_seats_1, avail_seats_2** : Both the functions work together to fetch various information related to the availability. They query on the views avail_seats_1_view and avail_seats_2_view respectively.
Feature Implemented : *Availability of seats*
- **checkpassword** : This function is responsible for checking the entered password when a user logs into the system. It queries from a materialized view.
Feature Implemented : *Login*
- **seatbook.class** : A total of 5 functions that are used to book the tickets for the respective classes. These are a part of the booking as well as cancellation

transactions and play the part of updating the availability tables for the particular operations.

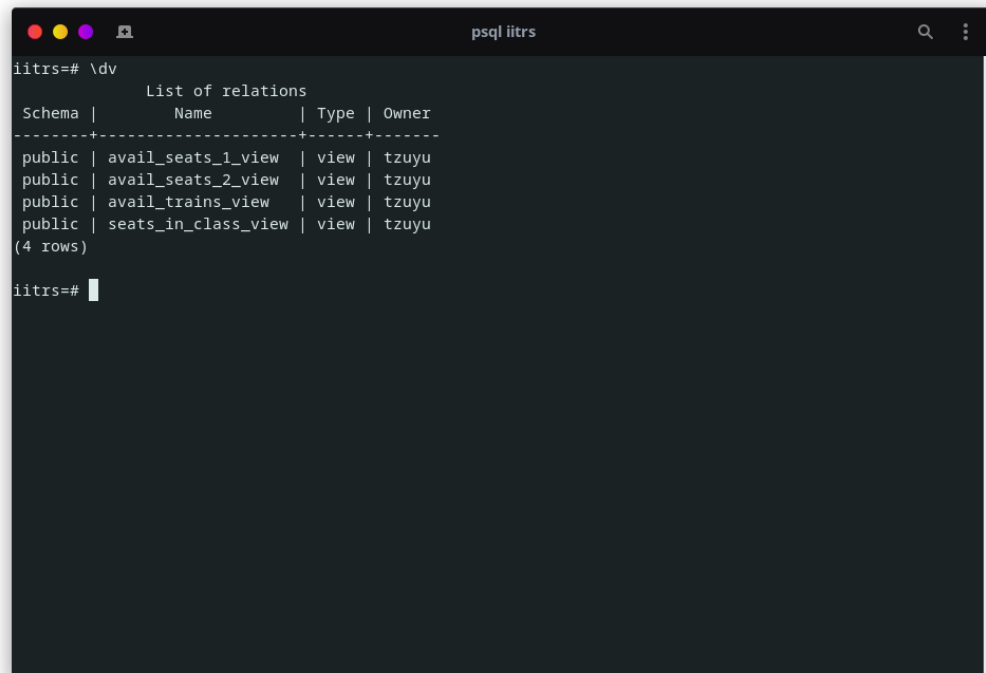
Feature Implemented : *Booking , Cancellation*

- **get_im_av_id** : This is one of the crucial functions that helps to update the availability of all the intermediate stations between the two stations for which the ticket is booked. This also helps in freeing up the seats and allocating them back to availability as a part of the cancellation transaction.

Feature Implemented : *Booking , Cancellation*

- **seats_in_class** : This is used while the manager adds new trains or routes or updates the availability. It adds the corresponding number of seats to the availability table for the number of coaches present in that train and the number of seats in the class.

Views



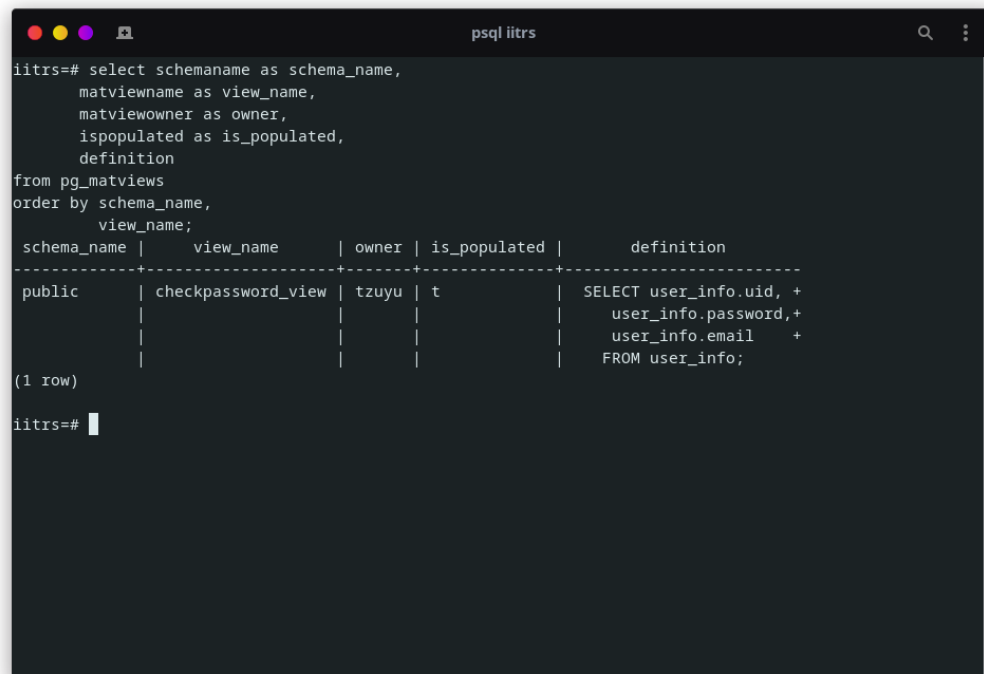
```
psql iitrs
iitrs=# \dv
          List of relations
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | avail_seats_1_view | view | tzuyu
public | avail_seats_2_view | view | tzuyu
public | avail_trains_view  | view | tzuyu
public | seats_in_class_view | view | tzuyu
(4 rows)

iitrs=#
```

Figure 10: Views in the database

The following views are implemented to increase the efficiency of the functionalities:

- **avail_seats_1_view, avail_seats_2_view** : These views are created for the purpose of security and are the only visible views to the Guest user. They are used while querying for the availability functions. Since it is one of the most frequent queries, we are using a view for this.
- **avail_trains_view** : This is used to look up for the available trains. Since it is one of the most frequent queries, we are using a view for this.
- **seats_in_class_view** : This view is generated since we need to query the join of train and the route table frequently and need to limit its access according to the roles too.
- **Materialized view, checkpassword_view** : Since this view would be less rarely updated but more frequently used, we use materialized views which result in a more better efficiency an faster operations as the data is stored in the memory.



```
iitrs=# select schemaname as schema_name,
      matviewname as view_name,
      matviewowner as owner,
      ispopulated as is_populated,
      definition
from pg_matviews
order by schema_name,
      view_name;
 schema_name | view_name | owner | is_populated | definition
-----+-----+-----+-----+-----
 public      | checkpassword_view | tzuyu | t             | SELECT user_info.uid, +
      |                  |      |               | user_info.password, +
      |                  |      |               | user_info.email  +
      |                  |      |               | FROM user_info;
(1 row)

iitrs=#
```

Figure 11: Material View

Triggers



```
psql iitrs
iitrs=# SELECT event_object_table AS table_name ,trigger_name
FROM information_schema.triggers
GROUP BY table_name , trigger_name
ORDER BY table_name ,trigger_name
;
 table_name |      trigger_name
-----
 ticket     | trg_chk_book_date
 user_info  | trg_refresh_chk_pwd_view
(2 rows)

iitrs=#
```

Figure 12: Triggers in the database

- **refresh_chk_pwd_view** : Even though it is updated less frequently, the materialized view needs to be updated each time a data is added to the user_info table. So we make a trigger that updates the data for this view on every insertion of a new user in the system.
- **chk_book_date** : This trigger is called whenever user tries to book a ticket and the booking date is beyond the booking period of 3 months. As a result this should throw an error, when booking date is three months more than the booking date.

Indices

The indices are introduced to optimize the various queries that we need to process in the implementation of the functionalities.

- Hash based index on train_no attribute in the train_journey table.

Query :

```
connection.execute(helper.psycop.text
('SELECT journey FROM train_journey WHERE train_no
= {}'.format(train_no)))
```


- Multi-column index on source and destination attributes of the route table.

Query :

```
SELECT uti FROM route
WHERE route.source = in_src
AND route.destination = in_dest
```

- Multi-column index on the email and password attributes of the user_info table.

Query :

```
SELECT * FROM user_info
WHERE email = uname AND pwd = password
```

- B-tree index on the email attribute of the user_info table.

Query :

```
psycop.db.execute_ddl_and_dml_commands
("SELECT name FROM user_info WHERE uid = {}".format(uid))
```

- B-tree index on each of the columns corresponding to each class in the availability table.

Query :

```
SELECT * FROM availability
WHERE availability.av_id = in_av_id
AND availability.general >= no_of_seats
```

```
SELECT * FROM availability
WHERE availability.av_id = in_av_id
AND availability.first_ac >= no_of_seats
```

- Hash based index on the PNR attribute of the ticket table.

Query :

```
connection.execute(helper.psycop.text(
"SELECT av_id FROM ticket WHERE pnr = '{}'.format(pnr)))
```

Final Product

Here are a few screenshots demonstrating our implemented system. Since we have a lot of functionalities implemented, only a few screenshots have been put here.

The Terminal based implementation of the User-Interface supports keyboard based operation (using TABs and Enter Key) as well as Mouse based operation (using Clicks and scrolls).

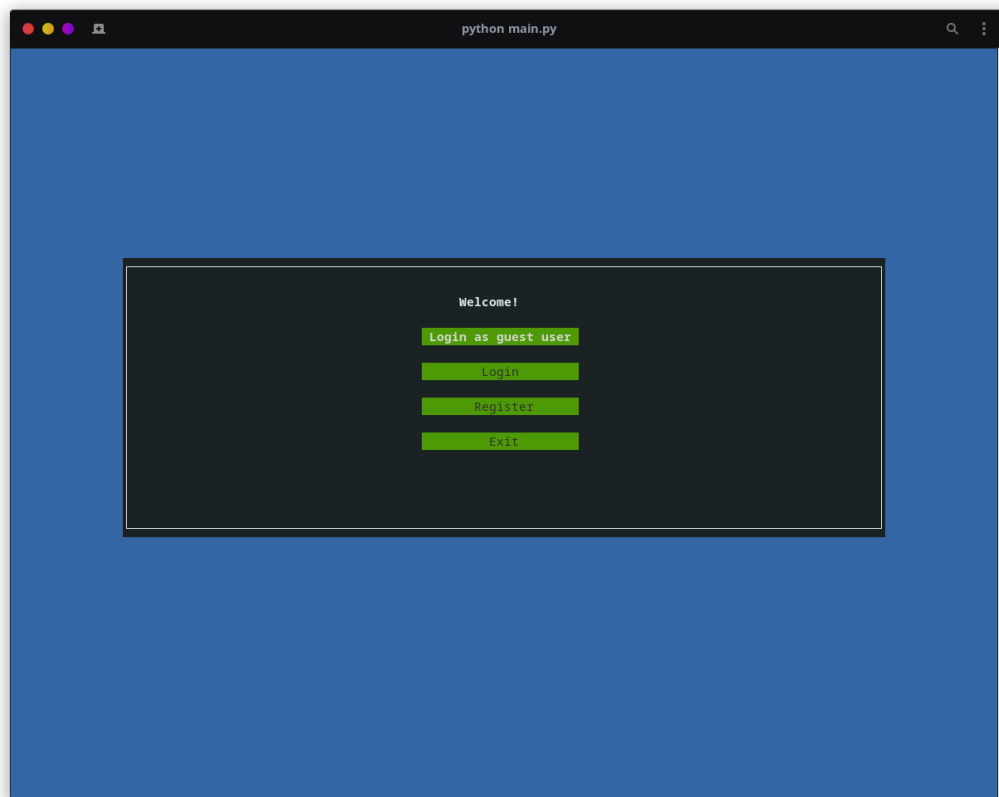


Figure 13:

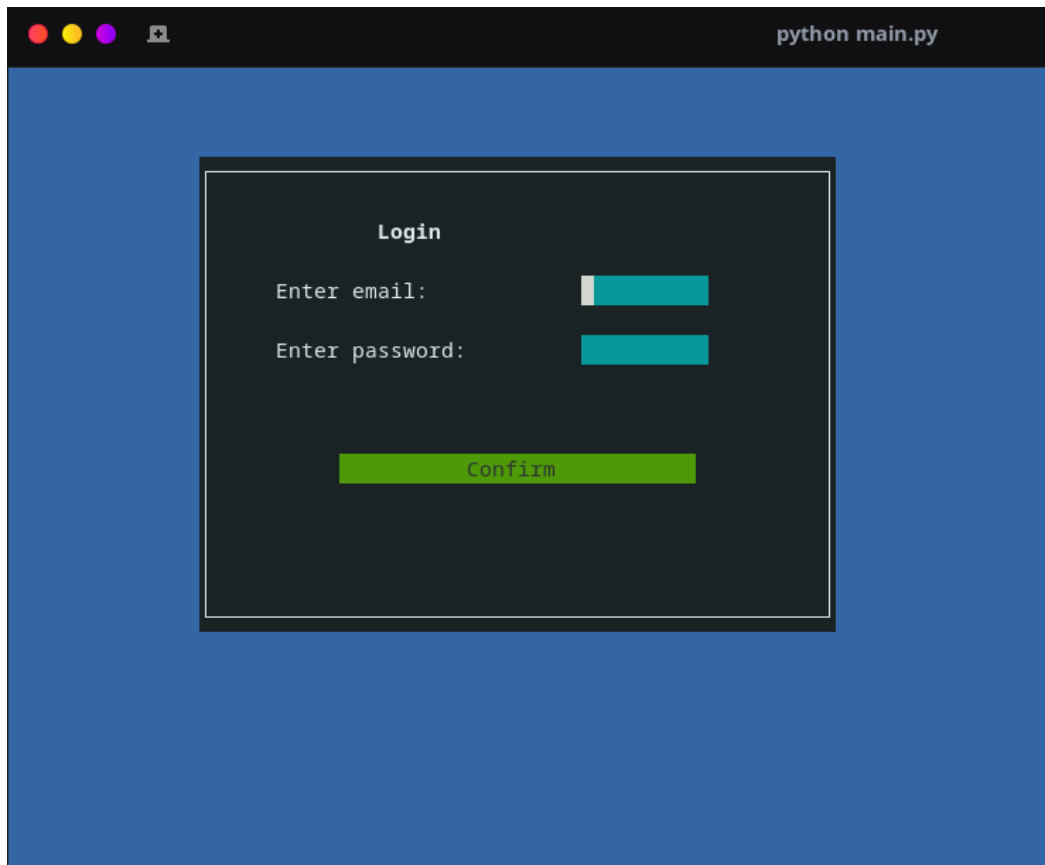


Figure 14:

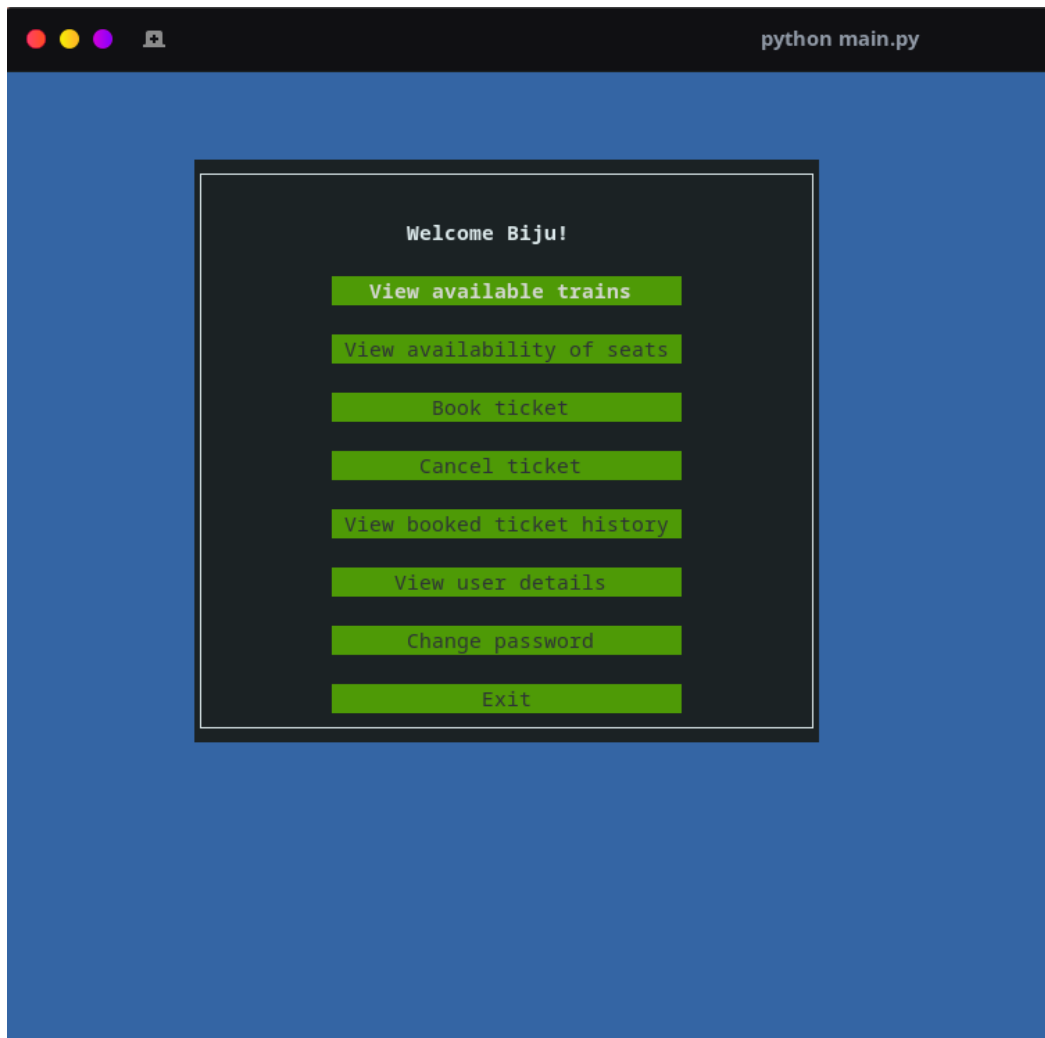


Figure 15:

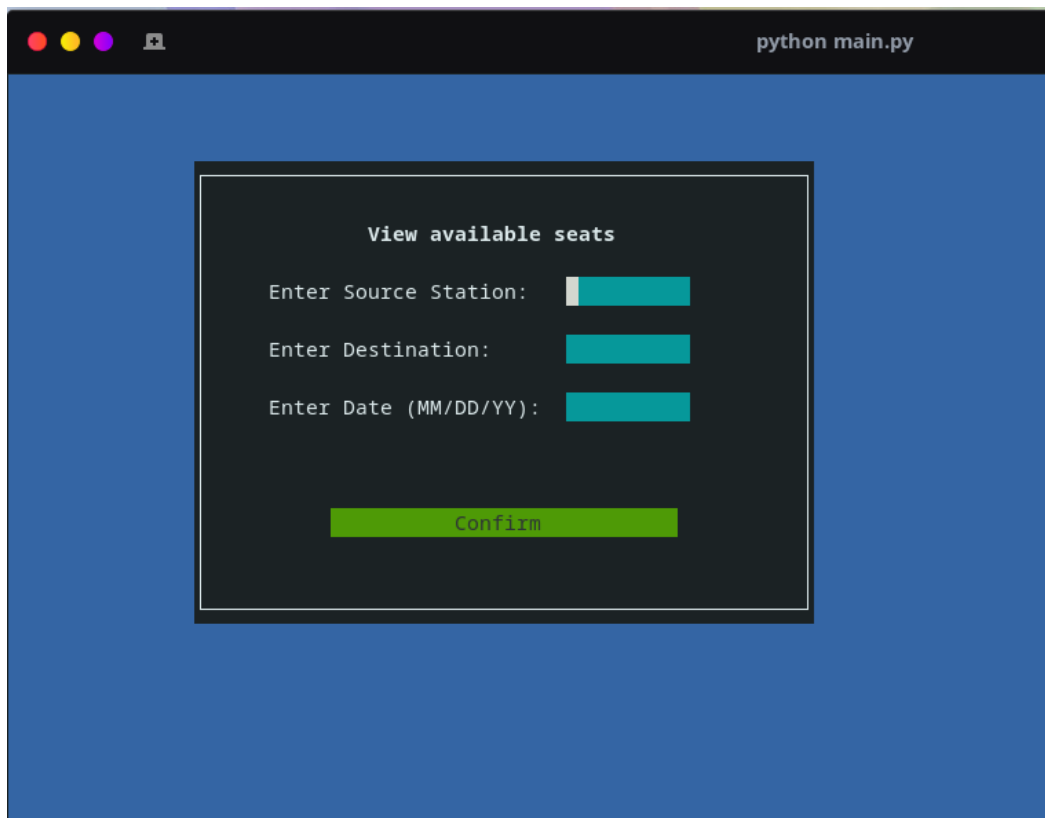


Figure 16:

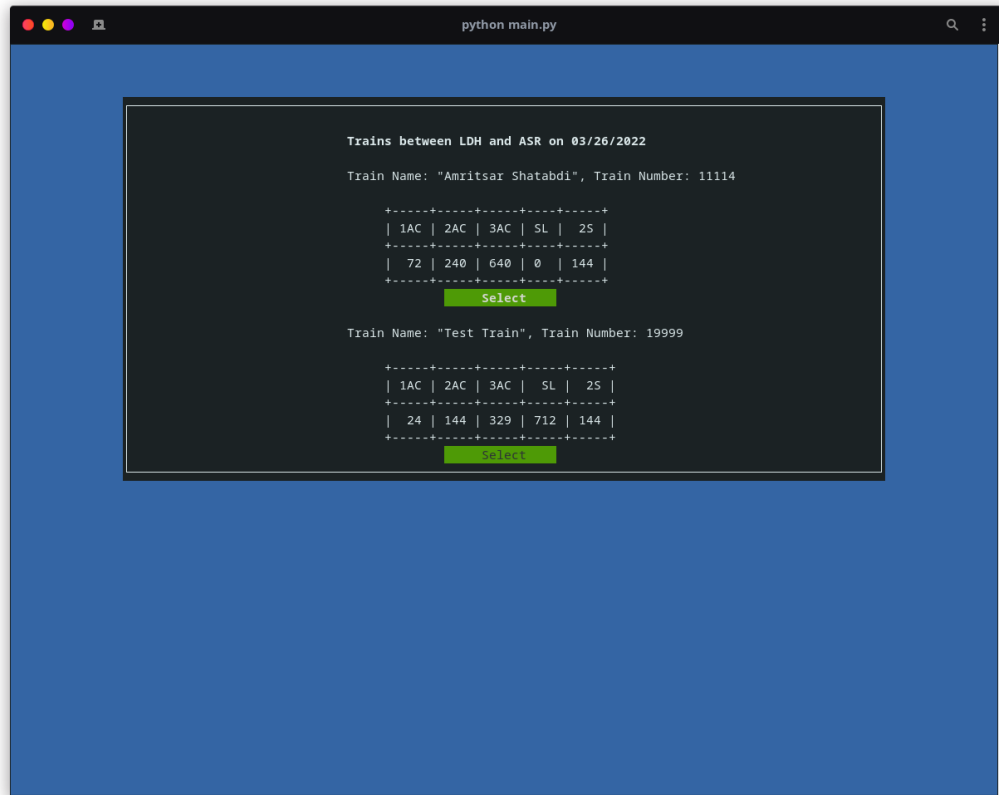


Figure 17:

Reproducing the project

To reproduce the project, first clone our github repository. Then first create a database named iitrs and dump iitrs.cas.sql into it. To run the program use `python main.py` (make sure your python version is up to date and is installed with the required libraries) For further clarification on installation, you can go through the README at the github page.