Team Members:
Asaduddin Ahmed 112201021
Vishal Rahangdale 112201049
Anup Kumar 112201042

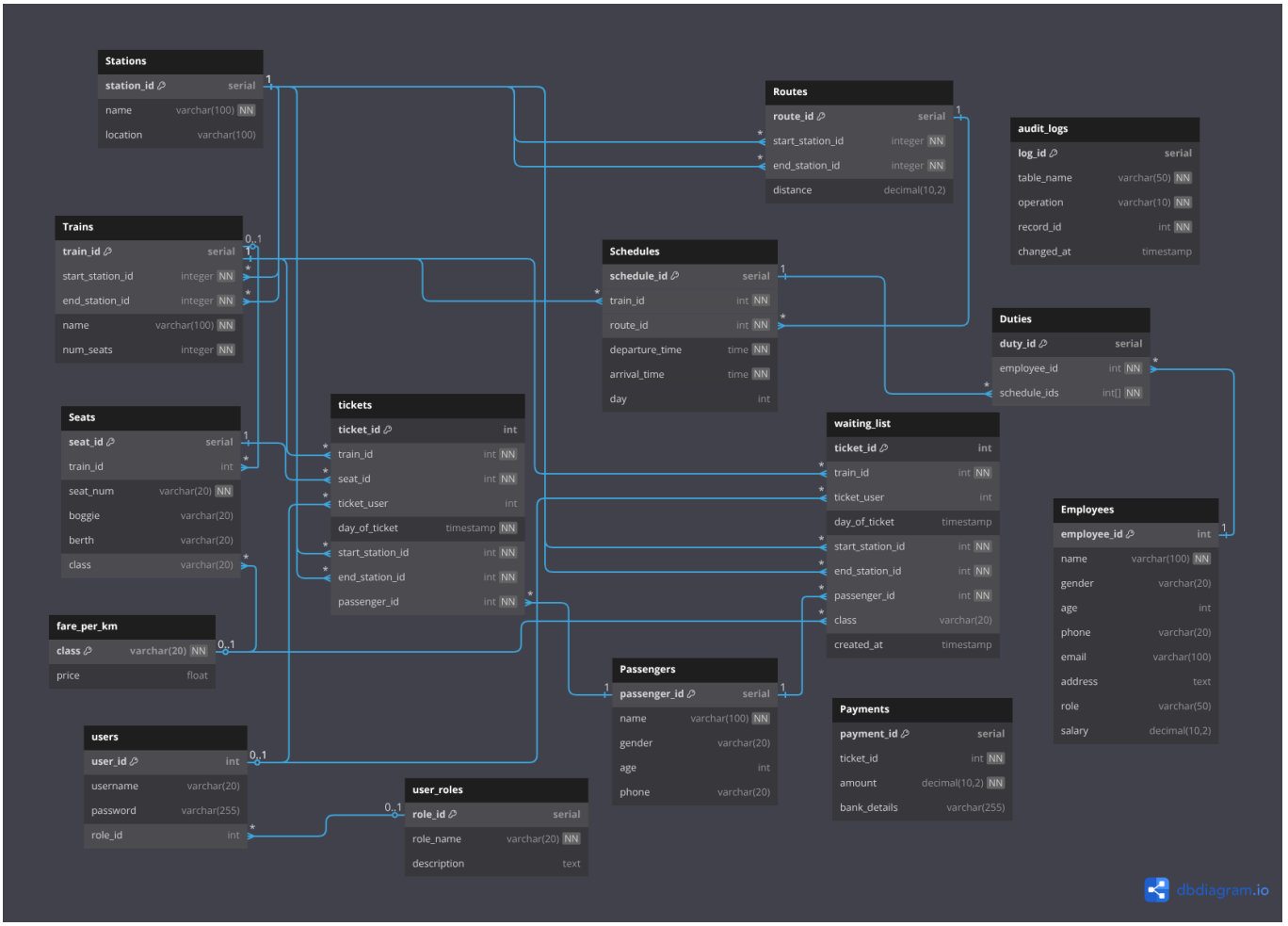
RAILWAY DATABASE MANAGEMENT SYSTEM:

Efficient railway management is crucial for ensuring smooth operations, passenger convenience, and optimal resource allocation. The Railway Management System (RMS) aims to streamline various aspects of railway operations, including train scheduling, ticket booking, seat allocation, employee management, fare calculations,waiting list ticket allocation after confirmed ticket cancellation.

The project consists of the following major components:

- **Relational Model:** Defines how data is structured across multiple tables, ensuring referential integrity.
- **Tables with Constraints:** Implements necessary constraints like **primary keys, foreign keys, uniqueness, and NOT NULL conditions** to maintain data consistency.
- **Implemented Functions ,Procedure and triggers**
- **Roles and Privileges**
- **Views and Indices**

Relational Model:



Tables with Constraints:

1.fare_per_km:

```
railway=# \d fare_per_km
                  Table "public.fare_per_km"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 class  | character varying(20) |           | not null |
 price  | double precision      |           |          |
Indexes:
    "fare_per_km_pkey" PRIMARY KEY, btree (class)
Referenced by:
    TABLE "seats" CONSTRAINT "seats_class_fkey" FOREIGN KEY (class) REFERENCES fare_per_km(class)
    TABLE "waiting_list" CONSTRAINT "waiting_list_class_fkey" FOREIGN KEY (class) REFERENCES fare_per_km(class)

railway=#
```

2.Stations:

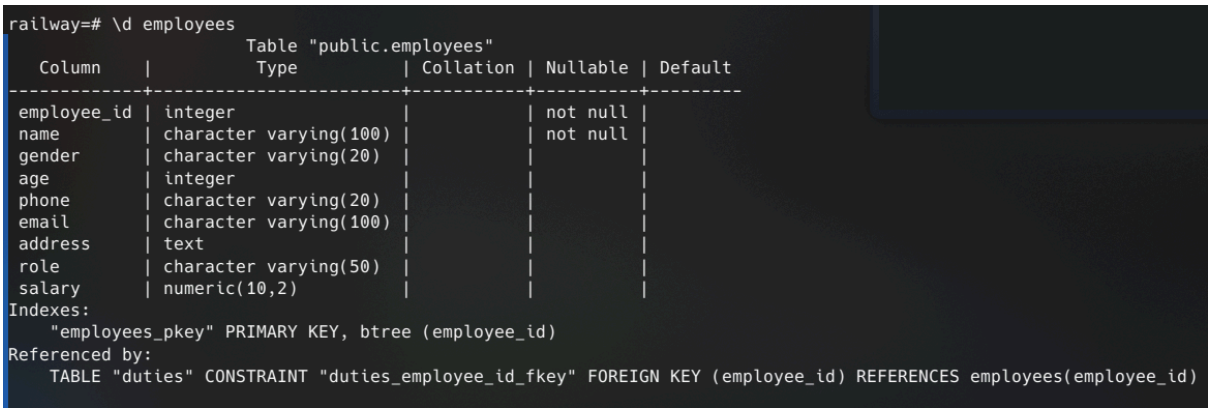
```
railway=# \d stations
                                      Table "public.stations"
   Column   |         Type          | Collation | Nullable |                  Default
------------+-----------------------+-----------+----------+-------------------------------------------
 station_id | integer               |           | not null | nextval('stations_station_id_seq'::regclass)
 name       | character varying(100) |           | not null |
 location   | character varying(100) |           |          |
Indexes:
    "stations_pkey" PRIMARY KEY, btree (station_id)
    "idx_stations_station_name" hash (name)
Referenced by:
    TABLE "routes" CONSTRAINT "routes_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    TABLE "routes" CONSTRAINT "routes_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
    TABLE "tickets" CONSTRAINT "tickets_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    TABLE "tickets" CONSTRAINT "tickets_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
    TABLE "trains" CONSTRAINT "trains_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    TABLE "trains" CONSTRAINT "trains_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
    TABLE "waiting_list" CONSTRAINT "waiting_list_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    TABLE "waiting_list" CONSTRAINT "waiting_list_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
```

3.Routes:

```
                                    Table "public.routes"
      Column      |     Type      | Collation | Nullable |                  Default
------------------+---------------+-----------+----------+-----------------------------------------
 route_id         | integer       |           | not null | nextval('routes_route_id_seq'::regclass)
 start_station_id | integer       |           | not null |
 end_station_id   | integer       |           | not null |
 distance         | numeric(10,2) |           |          |
Indexes:
    "routes_pkey" PRIMARY KEY, btree (route_id)
Foreign-key constraints:
    "routes_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    "routes_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
Referenced by:
    TABLE "schedules" CONSTRAINT "schedules_route_id_fkey" FOREIGN KEY (route_id) REFERENCES routes(route_id)
```

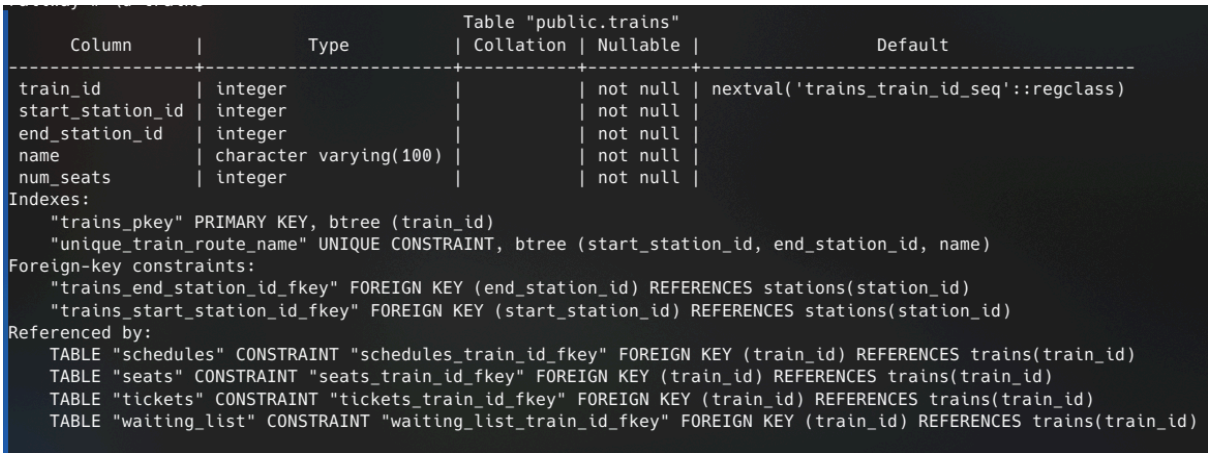
Since stations are from Stations table so from routes table start_station_id and end_station_id references that to maintain consistency.

4.Employees:



```
railway=# \d employees
                    Table "public.employees"
   Column    |          Type          | Collation | Nullable | Default
-------------+------------------------+-----------+----------+---------
 employee_id | integer                |           | not null |
 name        | character varying(100) |           | not null |
 gender      | character varying(20)  |           |          |
 age         | integer                |           |          |
 phone       | character varying(20)  |           |          |
 email       | character varying(100) |           |          |
 address     | text                   |           |          |
 role        | character varying(50)  |           |          |
 salary      | numeric(10,2)          |           |          |
Indexes:
    "employees_pkey" PRIMARY KEY, btree (employee_id)
Referenced by:
    TABLE "duties" CONSTRAINT "duties_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
```

4.Trains:

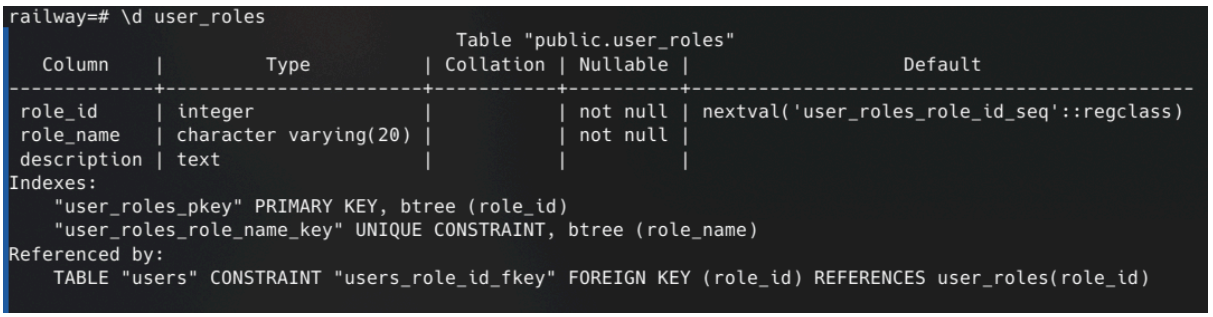


```
                                      Table "public.trains"
      Column      |          Type          | Collation | Nullable |                 Default
------------------+------------------------+-----------+----------+-----------------------------------------
 train_id         | integer                |           | not null | nextval('trains_train_id_seq'::regclass)
 start_station_id | integer                |           | not null |
 end_station_id   | integer                |           | not null |
 name             | character varying(100) |           | not null |
 num_seats        | integer                |           | not null |
Indexes:
    "trains_pkey" PRIMARY KEY, btree (train_id)
    "unique_train_route_name" UNIQUE CONSTRAINT, btree (start_station_id, end_station_id, name)
Foreign-key constraints:
    "trains_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    "trains_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
Referenced by:
    TABLE "schedules" CONSTRAINT "schedules_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
    TABLE "seats" CONSTRAINT "seats_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
    TABLE "tickets" CONSTRAINT "tickets_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
    TABLE "waiting_list" CONSTRAINT "waiting_list_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
```

To maintain consistency station_ids are referencing stations table.

There is an unique constraints which we put that a train can run between unique final stations.

5.user_roles:



```
railway=# \d user_roles
                                    Table "public.user_roles"
   Column    |          Type          | Collation | Nullable |                   Default
-------------+------------------------+-----------+----------+---------------------------------------------
 role_id     | integer                |           | not null | nextval('user_roles_role_id_seq'::regclass)
 role_name   | character varying(20)  |           | not null |
 description | text                   |           |          |
Indexes:
    "user_roles_pkey" PRIMARY KEY, btree (role_id)
    "user_roles_role_name_key" UNIQUE CONSTRAINT, btree (role_name)
Referenced by:
    TABLE "users" CONSTRAINT "users_role_id_fkey" FOREIGN KEY (role_id) REFERENCES user_roles(role_id)
```

We made role_name unique other wise with different role_id same role_name can be inserted which will create confusion with respect to user table which is referencing it.

6.users:

```
railway=# \d users
                    Table "public.users"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 user_id  | integer                |           | not null |
 username | character varying(20)  |           |          |
 password | character varying(255) |           |          |
 role_id  | integer                |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (user_id)
    "idx_users_username" hash (username)
    "users_username_key" UNIQUE CONSTRAINT, btree (username)
Foreign-key constraints:
    "users_role_id_fkey" FOREIGN KEY (role_id) REFERENCES user_roles(role_id)
Referenced by:
    TABLE "tickets" CONSTRAINT "tickets_ticket_user_fkey" FOREIGN KEY (ticket_user) REFERENCES users(user_id)
    TABLE "waiting_list" CONSTRAINT "waiting_list_ticket_user_fkey" FOREIGN KEY (ticket_user) REFERENCES users(user_id)
```

To authenticate users who are login we need this table.to maintain integral consistency it is referencing user_roles.

7.Schedules:

```
railway=# \d schedules
                                    Table "public.schedules"
     Column     |          Type          | Collation | Nullable |                    Default
----------------+------------------------+-----------+----------+-----------------------------------------------
 schedule_id    | integer                |           | not null | nextval('schedules_schedule_id_seq'::regclass)
 train_id       | integer                |           | not null |
 route_id       | integer                |           | not null |
 departure_time | time without time zone |           | not null |
 arrival_time   | time without time zone |           | not null |
 day            | integer                |           |          |
Indexes:
    "schedules_pkey" PRIMARY KEY, btree (schedule_id)
    "idx_schedules_route" btree (route_id)
    "idx_schedules_train" btree (train_id, day)
    "unique_schedule" UNIQUE CONSTRAINT, btree (train_id, route_id, departure_time, day)
Foreign-key constraints:
    "schedules_route_id_fkey" FOREIGN KEY (route_id) REFERENCES routes(route_id)
    "schedules_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
```

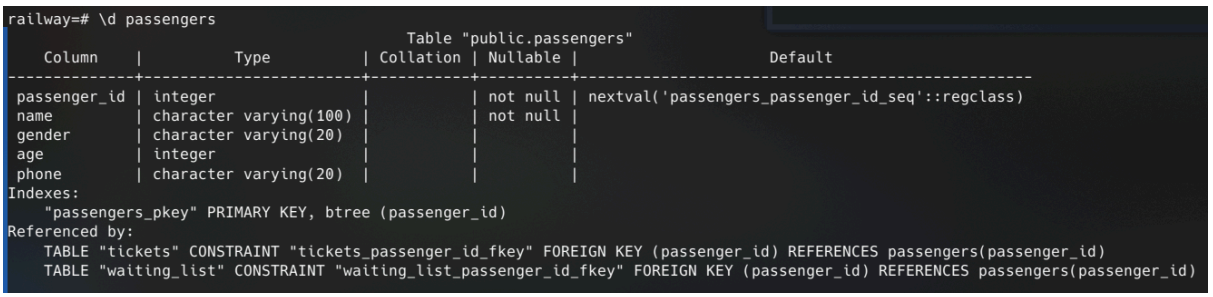
To maintain integrity trains and routes tables are referenced.here we put a unique constraint that for a particular route a train has a unique schedule.

8.Duties:

```
railway=# \d duties
                                Table "public.duties"
    Column    |   Type    | Collation | Nullable |                Default
--------------+-----------+-----------+----------+---------------------------------------
 duty_id      | integer   |           | not null | nextval('duties_duty_id_seq'::regclass)
 employee_id  | integer   |           | not null |
 schedule_ids | integer[] |           | not null |
Indexes:
    "duties_pkey" PRIMARY KEY, btree (duty_id)
    "unique_duty" UNIQUE CONSTRAINT, btree (employee_id)
Foreign-key constraints:
    "duties_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
```

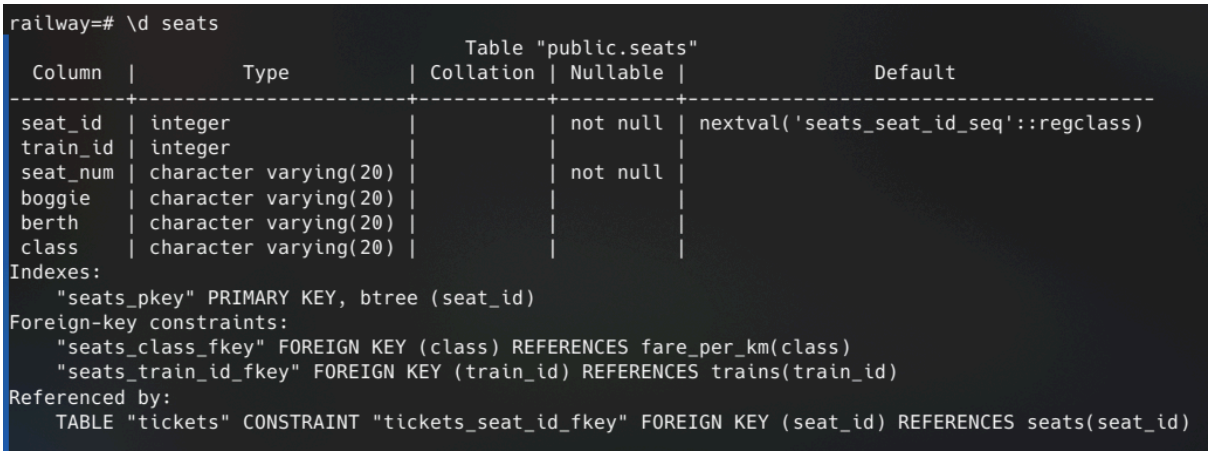
Each employee has a unique duties so we put a unique constraint on employee_id.To maintain integrity it is referencing the employee table.

9.Passengers:



```
railway=# \d passengers
                                        Table "public.passengers"
    Column    |          Type          | Collation | Nullable |                    Default
--------------+------------------------+-----------+----------+-----------------------------------------------
 passenger_id | integer                |           | not null | nextval('passengers_passenger_id_seq'::regclass)
 name         | character varying(100) |           | not null |
 gender       | character varying(20)  |           |          |
 age          | integer                |           |          |
 phone        | character varying(20)  |           |          |
Indexes:
    "passengers_pkey" PRIMARY KEY, btree (passenger_id)
Referenced by:
    TABLE "tickets" CONSTRAINT "tickets_passenger_id_fkey" FOREIGN KEY (passenger_id) REFERENCES passengers(passenger_id)
    TABLE "waiting_list" CONSTRAINT "waiting_list_passenger_id_fkey" FOREIGN KEY (passenger_id) REFERENCES passengers(passenger_id)
```

10.Seats:



```
railway=# \d seats
                                      Table "public.seats"
  Column   |          Type          | Collation | Nullable |                Default
-----------+------------------------+-----------+----------+---------------------------------------
 seat_id   | integer                |           | not null | nextval('seats_seat_id_seq'::regclass)
 train_id  | integer                |           |          |
 seat_num  | character varying(20)  |           | not null |
 boggie    | character varying(20)  |           |          |
 berth     | character varying(20)  |           |          |
 class     | character varying(20)  |           |          |
Indexes:
    "seats_pkey" PRIMARY KEY, btree (seat_id)
Foreign-key constraints:
    "seats_class_fkey" FOREIGN KEY (class) REFERENCES fare_per_km(class)
    "seats_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
Referenced by:
    TABLE "tickets" CONSTRAINT "tickets_seat_id_fkey" FOREIGN KEY (seat_id) REFERENCES seats(seat_id)
```

It have seats per train for each class so it to maintain integrity and consistency referencing trains and fare_per_km tables.

11.Tickets:

```
railway=# \d tickets
                                        Table "public.tickets"
      Column      |            Type             | Collation | Nullable |                Default
------------------+-----------------------------+-----------+----------+---------------------------------------
 ticket_id        | integer                     |           | not null | nextval('shared_ticket_id'::regclass)
 train_id         | integer                     |           | not null |
 seat_id          | integer                     |           | not null |
 ticket_user      | integer                     |           |          |
 day_of_ticket    | timestamp without time zone |           | not null |
 start_station_id | integer                     |           | not null |
 end_station_id   | integer                     |           | not null |
 passenger_id     | integer                     |           | not null |
Indexes:
    "tickets_pkey" PRIMARY KEY, btree (ticket_id)
    "idx_tickets_date" hash (day_of_ticket)
    "idx_tickets_passenger" btree (passenger_id)
    "idx_tickets_stations" btree (start_station_id, end_station_id)
    "idx_tickets_train_date" btree (train_id, day_of_ticket)
    "idx_tickets_user_id" hash (ticket_user)
    "unique_ticket" UNIQUE CONSTRAINT, btree (train_id, seat_id, day_of_ticket, passenger_id)
Foreign-key constraints:
    "tickets_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    "tickets_passenger_id_fkey" FOREIGN KEY (passenger_id) REFERENCES passengers(passenger_id)
    "tickets_seat_id_fkey" FOREIGN KEY (seat_id) REFERENCES seats(seat_id)
    "tickets_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
    "tickets_ticket_user_fkey" FOREIGN KEY (ticket_user) REFERENCES users(user_id)
    "tickets_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
```

To maintain integrity and consistency it is referencing trains,seats,stations and passengers tables. We put a unique constraint that for a passenger on a particular day for a particular train it has a unique ticket.

12.waiting_list:

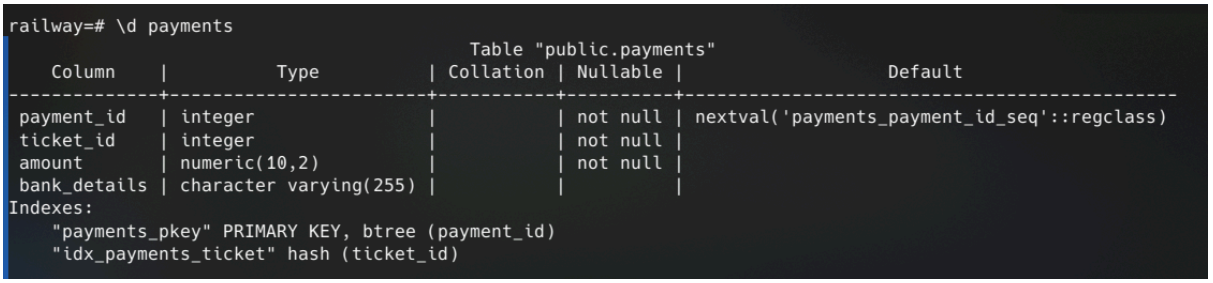
```
railway=# \d waiting_list
                                     Table "public.waiting_list"
      Column      |            Type             | Collation | Nullable |                Default
------------------+-----------------------------+-----------+----------+---------------------------------------
 ticket_id        | integer                     |           | not null | nextval('shared_ticket_id'::regclass)
 train_id         | integer                     |           | not null |
 ticket_user      | integer                     |           |          |
 day_of_ticket    | timestamp without time zone |           |          |
 start_station_id | integer                     |           | not null |
 end_station_id   | integer                     |           | not null |
 passenger_id     | integer                     |           | not null |
 class            | character varying(20)       |           |          |
 created_at       | timestamp without time zone |           |          | now()
Indexes:
    "waiting_list_pkey" PRIMARY KEY, btree (ticket_id)
    "idx_waitlist_allocation" btree (train_id, day_of_ticket)
    "idx_waitlist_date" hash (day_of_ticket)
    "idx_waitlist_user_id" hash (ticket_user)
    "unique_waiting_list" UNIQUE CONSTRAINT, btree (train_id, day_of_ticket, passenger_id)
Foreign-key constraints:
    "waiting_list_class_fkey" FOREIGN KEY (class) REFERENCES fare_per_km(class)
    "waiting_list_end_station_id_fkey" FOREIGN KEY (end_station_id) REFERENCES stations(station_id)
    "waiting_list_passenger_id_fkey" FOREIGN KEY (passenger_id) REFERENCES passengers(passenger_id)
    "waiting_list_start_station_id_fkey" FOREIGN KEY (start_station_id) REFERENCES stations(station_id)
    "waiting_list_ticket_user_fkey" FOREIGN KEY (ticket_user) REFERENCES users(user_id)
    "waiting_list_train_id_fkey" FOREIGN KEY (train_id) REFERENCES trains(train_id)
```

To maintain integrity and consistency it is referencing trains,seats,stations and passengers tables. We put a unique constraint that for a passenger on a particular day for a particular train it has a unique ticket.

```
create sequence shared_ticket_id start with 1 increment by 1;
```

We have made this shared_ticket_id between waiting_list and tickets. ensure unique ticket IDs across both tickets and waiting_list tables

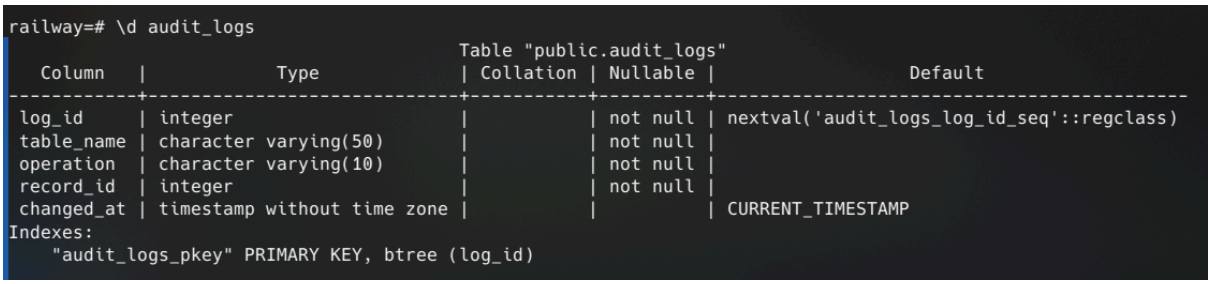
13.Payments:



```
railway=# \d payments
                                      Table "public.payments"
    Column    |          Type          | Collation | Nullable |                  Default
--------------+------------------------+-----------+----------+-------------------------------------------
 payment_id   | integer                |           | not null | nextval('payments_payment_id_seq'::regclass)
 ticket_id    | integer                |           | not null |
 amount       | numeric(10,2)          |           | not null |
 bank_details | character varying(255) |           |          |
Indexes:
    "payments_pkey" PRIMARY KEY, btree (payment_id)
    "idx_payments_ticket" hash (ticket_id)
```

Here to maintain integrity and consistency with respect to tickets which are either confirmed or in waiting list we have made trigger which before insert checking that upcoming tickets is at least present in one of tables tickets or waiting_list.Because we know a column can't reference two tables at a time.

14.audit_logs:



```
railway=# \d audit_logs
                                    Table "public.audit_logs"
   Column   |            Type             | Collation | Nullable |                  Default
------------+-----------------------------+-----------+----------+-------------------------------------------
 log_id     | integer                     |           | not null | nextval('audit_logs_log_id_seq'::regclass)
 table_name | character varying(50)       |           | not null |
 operation  | character varying(10)       |           | not null |
 record_id  | integer                     |           | not null |
 changed_at | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "audit_logs_pkey" PRIMARY KEY, btree (log_id)
```

Implemented Functions ,Procedure and triggers:

A.Functions and Procedures:

1.allot:

This function is used while booking the tickets for passengers.

This prevent the double booking of Seats.

2.book_ticket:

This function is booking ticket by using allot function.

Ensures that only valid stations are used by checking their existence.

3.cancel_ticket:

It deletes the ticket of a user for given pnr from the tickets table if the ticket is confirmed otherwise from the waiting_list.

When a confirmed ticket is deleted two triggers occur which one checks if a waiting_list passenger can be confirmed or not and another trigger is deleting deleted passenger's ticket payments from payment table and details from passengers table.

So:

- Delegates waitlist seat reallocation to triggers for modularity and reliability.
- Maintains referential integrity by also cleaning up related payment and passenger data.
- Prevents unauthorized cancellations by enforcing ticket-user matching.

4.deletetickets:

This function is used by an event to delete all expired tickets(tickets for which journey is completed) from tickets table

and from waiting_list table.and also deleting related information from payments and passenger tables.

So:

- Ensures the system does not retain outdated ticket records,payments and passenger details keeping the database lean.
- Prevents buildup of stale records that could interfere with allocation logic or reporting.

5.get_amount:

This is helper function used by a function pay for getting the total amount of a ticket.

6.get_route:

This is helper function used by allot to get route of a train for ticket booking.

7.insert_seats:

This function is inserting the seats for a train in seat table per class per boggie.

The count of seats inserted for a train is not greater than the total available seats in train.

8.pnr_status:

It takes the pnr and give whether ticket is confirmed or not.

9.register_user:

This function is used for new user registration.it is ensuring that username should be unique.

- Prevents duplicate usernames – a crucial constraint for authentication.
- Guarantees consistent role assignment through referential integrity (role_id FK to user_roles).

10.search_trains:

This function gives trains on particular date for particular class(default general) between two stations with available seats,timing and total cost of ticket.

11.delete_expired_ticket:

This is a function (trigger like) a pg_cron based event(scheduled at midnight). It invoked at midnight and then checks all train whose journey is completed and then by using deletetickets function it deletes all corresponding tickets.

12.add_ticket_on_cancel:

This function is invoked by a trigger after delete on the ticket table and after that it checks if a waiting_list ticket can be confirmed.

B.Triggers:

1.ticket_cancel_trigger:

As mentioned above after ticket is deleted it triggers and calling function add_ticket_on_cancel as mentioned above that check any waiting_list passenger can be given a confirm ticket.

2.log_changes:

Whenever ticket table have insert,update and delete it triggers and store operations and time.

3.validate_ticket_id_exists:

This triggers before insert on payments as mentioned earlier,it check whether the ticket is present at least one of table tickets or in waiting_list.

4.delete_ticket_info:

This triggers after delete on ticket table so as a passenger ticket is deleted its payment and passenger's details should be deleted and this trigger is doing that.

Roles and list of privileges:

We have three group roles:

1)admin_role:

Complete privilege of railway database.

2)employee_role:

Privileges:

select on fare_per_km,Stations,Routes,Employees,users,user_role

s,Trains,Schedules,Duties,Passengers,Seats,tickets,waiting_list,payments,audit_logs

insert on Passengers,tickets,waiting_list,

users,payments,audit_logs

delete on Passengers,tickets,waiting_list,Payments,

audit_logs

USAGE, SELECT ON SEQUENCE
passengers_passenger_id_seq,audit_logs_log_id_seq,shared_ticket_id,payments_payment_id_seq

For function execution:register_user,pnr_status,search_trains,cancel_ticket,get_user_tickets,book_ticket,allot,user_bookings,employee_duties,get_route.

3)user_role:

Privileges:

select on fare_per_km,Stations,Routes,users,user_roles,Trains,Schedules,Duties,Passengers,Seats,tickets,waiting_list,payments,audit_logs

insert on Passengers,tickets,waiting_list,

users,payments,audit_logs

delete on Passengers,tickets,waiting_list,Payments,

audit_logs

USAGE, SELECT ON SEQUENCE
passengers_passenger_id_seq,audit_logs_log_id_seq,shared_ticket_id,payments_payment_id_seq

For function execution:register_user,pnr_status,search_trains,cancel_ticket,get_user_tickets,book_ticket,allot,user_bookings,get_route.

Views:

1.admin_train_overview

Admin view for trains with route ,confirmed tickets count and waiting_list count information.

Admin needs to know which trains are there what are there routes, they are in heavy use(more travellers) or normal.

2.admin_duties_view

Admin needs to know which employee is assigned duties on which trains.

3.employee_duties

This is a parameterized view that takes logged in employee_id and shows her/his duty schedule with trains name and timing.

It facilitates employees to easily know updated duties.

4.user_bookings

This is also a parameterised view that takes logged in user_id and give all ticket bookings done by that user confirmed or waiting_list.

Indexing:

On tickets table:

```
CREATE INDEX idx_tickets_user_id ON tickets using HASH(ticket_user);

---for status of ticket requested by user in user_bookings and in pnr_status

CREATE INDEX idx_tickets_date ON tickets using HASH(day_of_ticket);

-- for event based deletion of tickets on midnight

CREATE INDEX idx_tickets_train_date ON tickets(train_id, day_of_ticket);

-- allot and search_trains function

CREATE INDEX idx_tickets_stations ON tickets(start_station_id, end_station_id);
```

```
-- need to check how many people are travelling
between 2 station by admin

CREATE INDEX idx_tickets_passenger ON
tickets(passenger_id);

-- to finding user tickets
```

On waiting_list:

```
CREATE INDEX idx_waitlist_user_id ON waiting_list
using HASH(ticket_user);

-- for status of ticket requested by user in
user_bookings and in pnr_status

CREATE INDEX idx_waitlist_date ON waiting_list using
HASH(day_of_ticket);

-- for event based deletion of tickets on midnight

CREATE INDEX idx_waitlist_allocation ON
waiting_list(train_id,day_of_ticket);
--add_ticket_on_cancel trigger uses this for
efficient lookups of waiting list
```

On Schedules:

```
CREATE INDEX idx_schedules_train ON
schedules(train_id, day);
-- for get_route function

CREATE INDEX idx_schedules_route ON
schedules(route_id);
-- need to check how many trains pass through this
route
```

On users:

```
CREATE INDEX idx_users_username ON users using
HASH(username);

-- for efficient lookups during login
```

On payments:

```
CREATE INDEX idx_payments_ticket ON payments using
HASH(ticket_id);

-- for deletion of payments table given ticket_id on
ticket cancel
```

On Stations:

```
CREATE INDEX idx_stations_station_name on Stations
using HASH(name);

--- used by search_trains function for finding
station_id of given station_name.
```