Mid Term Project Report

CS3120 - Database Management Systems Laboratory

Railway Reservation System

Group 5:
111901005 Aditya Agarwal
111901030 Mayank Singla
111901058 Satyam Mishra

Introduction

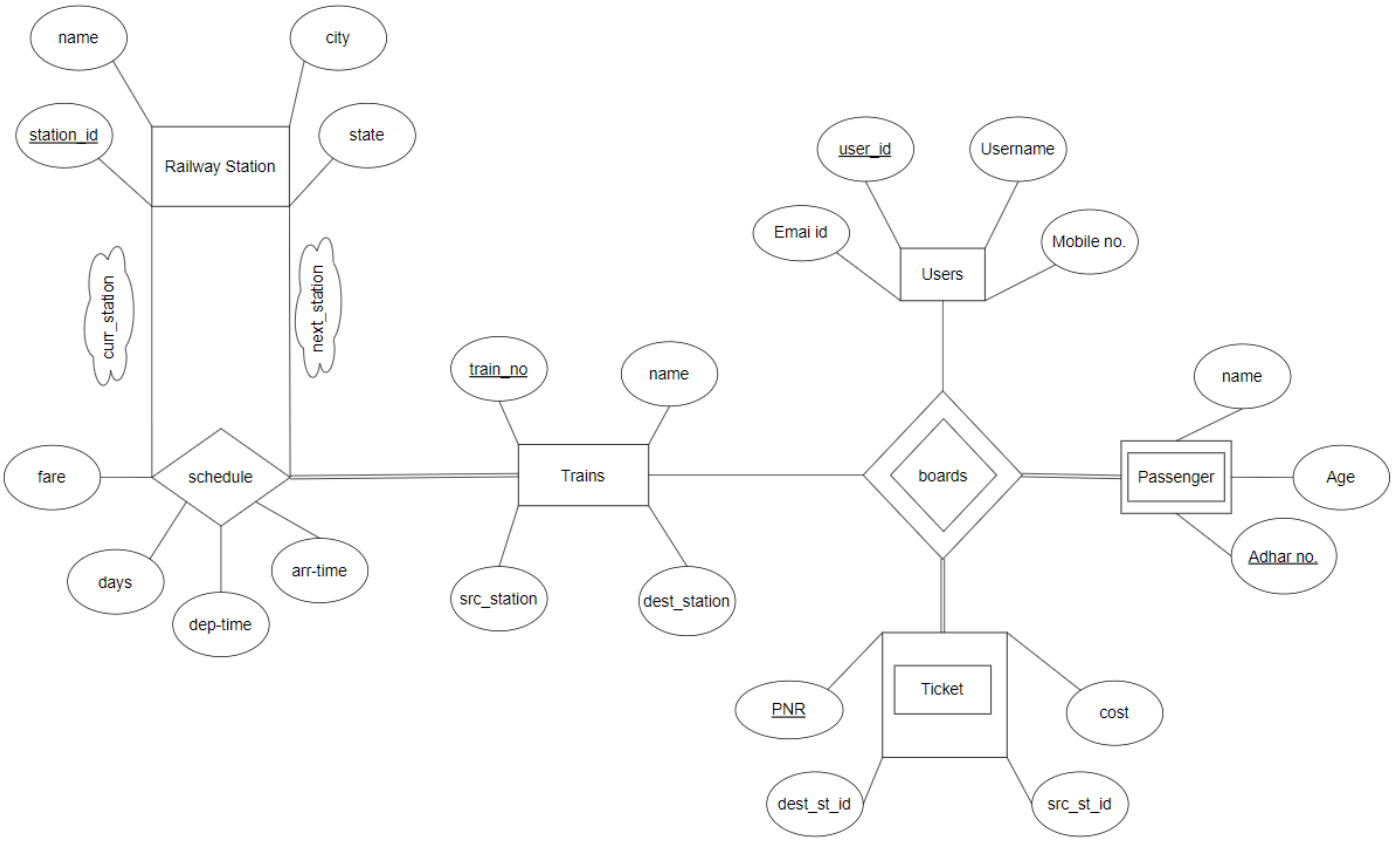
This project aims to build a railway reservation system which is a software application that handles the entire booking data of the railway. It has been developed to override the problems prevailing in the practical manual system. It is fully based on the concept of reserving train tickets for various destinations reliably and consistently. It reduces the stress and workload of the employee who books the ticket. It also lets the travelers book and finds schedules for the trains with ease. This software can also be used by different railway companies to carry out operations in a smooth, effective, and automated manner.

Features of the Project

- Maintaining separate logins for the passengers as well as admin. The admin login will be via a special E-mail ID given to him by the railway authorities.
- Only admin can create, update and delete the trains.
- The passengers should be able to see the actual data of available trains for some given source and destination city.
- The passengers should be able to see the schedule of a particular train on some particular day and also the schedule of all the trains at a particular station.
- The passenger should be able to book as well as cancel a ticket for a train.
- Manage the information of the ticket and we should be able to extract out the details of a train as well as customer associated with a given PNR number of a ticket.
- There is a user who could also book a ticket not only for himself but for other passengers who might not be the users of the system.

- It should avoid errors while entering the data and provide suitable error messages while entering invalid data.

ER Diagram



Entities

- Railway Station
- Trains
- Users
- Passengers
- Ticket

Relations

- Schedule
- Boards

Structure of the Database and Functionality

Here we are displaying the tables and their constraints in form of PostgreSQL description of tables to display the results better. In the above ER diagram **board** is a weak relation and hence its table will not be required as it will be redundant.

Users

```
                                    Table "public.users"
  Column   |          Type          | Collation | Nullable |               Default
-----------+------------------------+-----------+----------+--------------------------------------
 user_id   | integer                |           | not null | nextval('users_user_id_seq'::regclass)
 username  | character varying(100) |           | not null |
 emailid   | character varying(500) |           | not null |
 mobile_no | character varying(20)  |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (user_id)
Check constraints:
    "users_emailid_check" CHECK (emailid::text ~ similar_escape('^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'::text, NULL::text))
Referenced by:
    TABLE "passenger" CONSTRAINT "passenger_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
    TABLE "ticket" CONSTRAINT "ticket_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
```

This table will contain the users of the table i.e. those who can book tickets and those who can update the railway data. This table contains `user_id` which is the primary key for the table. We have a username, email id, and mobile number as well. Here we have added a check over the email id to follow a regular expression for a valid email id. All the other constraints implemented on other columns are taken from the PostgreSQL inbuilt constraints.

We will have the role of the **Guest user** for whom we won't be storing any data, but it will be only able to read the data and do queries on our database. (As if a guest user will do on the websites of a real railway management company). This role will hence only have read-only permissions to the database.

We will have another role of **Passenger** who is a traveler. We will be only storing some data of this role to keep track of it on the ticket. This user will have no permission to access our database other than the permissions of the guest user. Like in a real system, there will be an authenticated user who can book tickets for other passengers as well and those passengers need not be a user. That authenticated user is tracked in our database with the role of a **User** for whom we will be storing its login credentials and it will be given permissions to book a ticket (insert/update/delete) for a train from some source to destination and also all the permissions of the guest user.

Lastly, we will be having an **Admin** role who will be responsible for managing the railway data and will have all the permissions on every object of the database. This role will also be having some login credentials that will be tracked using a special E-mail id

that will be assigned to it via the railway authorities (or he if he is the only admin and the owner of the database).

Passenger

```
                                    Table "public.passenger"
  Column  |          Type          | Collation | Nullable |                    Default
----------+------------------------+-----------+----------+-----------------------------------------------
 aadhar   | uuid                   |           | not null |
 age      | integer                |           | not null |
 name     | character varying(225) |           | not null |
 train_no | integer                |           | not null | nextval('passenger_train_no_seq'::regclass)
 user_id  | integer                |           | not null |
Indexes:
    "passenger_pkey" PRIMARY KEY, btree (aadhar)
Check constraints:
    "passenger_age_check" CHECK (age > 0)
Foreign-key constraints:
    "passenger_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
    "passenger_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
```

This table will contain the details of a passenger. A passenger need not be a user. But in our system, if someone wants to book a ticket then he needs to be a user of the database. Here `aadhar` is the primary key for the passenger. We even have `age`, `name`, `train_no`, `user_id` as attributes of this table. This entity is a weak entity as it has no sense of its own. All the constraints implemented on other columns are taken from the PostgreSQL inbuilt constraints.

Railway Stations

```
                                  Table "public.railway_station"
   Column   |          Type          | Collation | Nullable |                      Default
------------+------------------------+-----------+----------+-------------------------------------------------
 station_id | integer                |           | not null | nextval('railway_station_station_id_seq'::regclass)
 city       | character varying(50)  |           | not null |
 state      | character varying(50)  |           | not null |
 name       | character varying(100) |           | not null |
Indexes:
    "railway_station_pkey" PRIMARY KEY, btree (station_id)
Referenced by:
    TABLE "schedule" CONSTRAINT "schedule_curr_station_id_fkey" FOREIGN KEY (curr_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    TABLE "schedule" CONSTRAINT "schedule_next_station_id_fkey" FOREIGN KEY (next_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    TABLE "ticket" CONSTRAINT "ticket_dest_station_id_fkey" FOREIGN KEY (dest_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    TABLE "ticket" CONSTRAINT "ticket_src_station_id_fkey" FOREIGN KEY (src_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    TABLE "train" CONSTRAINT "train_dest_station_fkey" FOREIGN KEY (dest_station) REFERENCES railway_station(station_id) ON DELETE CASCADE
    TABLE "train" CONSTRAINT "train_src_station_fkey" FOREIGN KEY (src_station) REFERENCES railway_station(station_id) ON DELETE CASCADE
```

This table has information about all the railway stations that are there in the database. We have `station_id` as the primary key for the station. It even has the city, state, and name of the station. All the constraints implemented on other columns are taken from the PostgreSQL inbuilt constraints.

Trains

```
                        Table "public.train"
    Column     |          Type          | Collation | Nullable | Default
---------------+------------------------+-----------+----------+---------
 train_no      | integer                |           | not null |
 name          | character varying(500) |           | not null |
 src_station   | integer                |           | not null |
 dest_station  | integer                |           | not null |
Indexes:
    "train_pkey" PRIMARY KEY, btree (train_no)
Check constraints:
    "train_train_no_check" CHECK (train_no > 0)
Foreign-key constraints:
    "train_dest_station_fkey" FOREIGN KEY (dest_station) REFERENCES railway_station(station_id) ON DELETE CASCADE
    "train_src_station_fkey" FOREIGN KEY (src_station) REFERENCES railway_station(station_id) ON DELETE CASCADE
Referenced by:
    TABLE "passenger" CONSTRAINT "passenger_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
    TABLE "schedule" CONSTRAINT "schedule_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
    TABLE "ticket" CONSTRAINT "ticket_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
```

This table contains the train number and train name. Here the `train_no` is the primary key. We have even stored the station id of the station where the train starts and the destination station where the whole journey of the train will end. All the constraints implemented on other columns are taken from the PostgreSQL inbuilt constraints.

Ticket

```
                        Table "public.ticket"
    Column      |  Type   | Collation | Nullable |      Default
----------------+---------+-----------+----------+--------------------
 pnr            | uuid    |           | not null | uuid_generate_v4()
 cost           | integer |           | not null |
 src_station_id | integer |           | not null |
 dest_station_id| integer |           | not null |
 train_no       | integer |           | not null |
 user_id        | integer |           | not null |
 aadhar         | uuid    |           | not null |
Indexes:
    "ticket_pkey" PRIMARY KEY, btree (pnr)
Check constraints:
    "ticket_cost_check" CHECK (cost > 0)
    "ticket_train_no_check" CHECK (train_no > 0)
Foreign-key constraints:
    "ticket_aadhar_fkey" FOREIGN KEY (aadhar) REFERENCES passenger(aadhar) ON DELETE CASCADE
    "ticket_dest_station_id_fkey" FOREIGN KEY (dest_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    "ticket_src_station_id_fkey" FOREIGN KEY (src_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    "ticket_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
    "ticket_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
```

This table contains the details of the ticket booked by a user. It has `PNR` as its primary key. It even has the `cost` of the journey and the source and destination station ids for the journey and the `aadhar` number of the passenger traveling through this ticket. This entity is a weak entity as it has no sense of its own.

Schedule

```
                                    Table "public.schedule"
     Column      |          Type           | Collation | Nullable |              Default
-----------------+-------------------------+-----------+----------+-----------------------------------
 id              | integer                 |           | not null | nextval('schedule_id_seq'::regclass)
 train_no        | integer                 |           | not null |
 curr_station_id | integer                 |           | not null |
 next_station_id | integer                 |           |          |
 days            | character varying(50)[] |           | not null |
 arr_time        | time without time zone  |           | not null |
 dep_time        | time without time zone  |           | not null |
 fare            | numeric(7,2)            |           | not null |
Indexes:
    "schedule_pkey" PRIMARY KEY, btree (id)
Check constraints:
    "schedule_check" CHECK (dep_time >= arr_time)
    "schedule_days_check" CHECK (days <@ ARRAY['Monday'::character varying, 'Tuesday'::character varying, 'Wednesday'::character varying,
 'Thursday'::character varying, 'Friday'::character varying, 'Saturday'::character varying, 'Sunday'::character varying])
    "schedule_fare_check" CHECK (fare >= 0::numeric)
Foreign-key constraints:
    "schedule_curr_station_id_fkey" FOREIGN KEY (curr_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    "schedule_next_station_id_fkey" FOREIGN KEY (next_station_id) REFERENCES railway_station(station_id) ON DELETE CASCADE
    "schedule_train_no_fkey" FOREIGN KEY (train_no) REFERENCES train(train_no) ON DELETE CASCADE
```

The schedule is a relation connecting railway stations and trains. In this table, for every train, we have a pair of station IDs and intermediate stations that lie on the route of the train from the source station to the destination station. Each entry of the table has the current station and the next station. This next station value will be NULL for the destination station. This relation has fare, days, dept-time, arr-time as attributes. In fare, we expect to get the fare from the source station to the current station of the train and it will always be a positive value. Days will tell the running days of the train and their values must be a subset of the standard days (that has been implemented via subset operator in PostgreSQL) and the arrival and departure times will tell the arrival and departure time for the current station and train respectively. All the other constraints implemented on other columns are taken from the PostgreSQL inbuilt constraints.

Assumptions

While designing the database, these are the assumptions that we have assumed for now and will slowly improve upon them as we progress and improvise our database management.

- We have an infinite number of platforms on a station.
- A train has infinite numbers of seats.
- While adding the train details the admin will provide us with the train schedule in sorted order.
- The ticket is for individual passengers only.
- A train will follow the same schedule on all its days.

Implementation of Constraints

Detailed information about the constraints on the individual columns has already been discussed above when explaining the structure of the database and tables.
Also, for each operation, we are making a procedure so that we will only use that procedure to do any query on the database. The procedures are implemented using transactions and error handling for the input validation is done by the methods and utilities provided by the **PL/pgSQL** wherever required to meet the requirements and for a better user experience.

Here is an example screenshot of one of the procedures that we implemented. We have provided a link to our codebase at the end.

```
CREATE TYPE schedule_info AS (
  days VARCHAR[],
  arr_time TIME,
  dep_time TIME,
  station_id INT,
  fare INT
);


CREATE OR REPLACE PROCEDURE insert_train(in_train_no INT, in_name VARCHAR, in_sch schedule_info[])
LANGUAGE PLPGSQL
AS $$
DECLARE
    x INT := array_length(in_sch, 1);
BEGIN WITH
    IF NOT (days <@ ARRAY['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']) THEN
    RAISE EXCEPTION 'Invalid Day'
    END IF;

    INSERT INTO trains(train_no, name)
    VALUES (in_train_no, in_name);

    FOR i IN 1..(x-1)
    LOOP
        INSERT INTO schedule(train_no, curr_station_id, next_station_id, arr_time, dep_time, fare, days)
        VALUES (in_train_no, in_sch[i].station_id, in_sch[i+1].station_id, in_sch[i].arr_time, in_sch[i].dep_time, in_sch[i].fare, in_sch[i].days)
    END LOOP;

    INSERT INTO schedule(train_no, curr_station_id, next_station_id, arr_time, dep_time, fare, days)
    VALUES (in_train_no, in_sch[x].station_id, NULL, in_sch[x].arr_time, in_sch[x].dep_time, in_sch[x].fare, in_sch[x].days);

    COMMIT;
END;
$$;
```

Procedure to insert data with checking constraints

Functionalities of the Project

The project is still under the development and testing phase. As of now, we support the following queries on the database.

a) **Give all trains for given source and destination and date.**
   Here we the user will enter the source and destination and city and the date of travel, we will handle it in the following way:
   1. Will capture all the stations present in the given source city then corresponding to these source stations we will find the train number in the schedule table.
   2. Corresponding to the above train number we will then find if there exists any station that belongs to the destination city and where the train arrives after it departs from the source station (time constraint).
   3. We will show all the trains which satisfy both the above conditions.

b) **Given a train, give the schedule for it on a day.**
   Using the trains table we will first get the source station for the given train. Then we can query on the schedule table and search for the row having the give station number as the station number and the current station as the found source station. There will be only one such row. We can use this row to the station following the source station. Now we can do the same with this station to get the next station. We will keep doing this until we don't get the next station as NULL. This station will be our destination station. We can extract all other information like arrival and departure times from the rows of the table.

c) **Give the schedule of trains at a given station.**
   We first need to query on the schedule table to get all the entries having the given station as the current station. Then we just need to sort all the entries based on the arrival time and the resulting table will give the schedule of all the trains that will be passing through the given station.

d) **For a given PNR, give details of the journey and the passenger.**
   Using the PNR and the tickets table, we can get the Aadhar number of the passenger corresponding to the given PNR, from which we can get all the details of the passenger. Similarly, the ticket table also contains the train number, source station column, destination station column which provides details whole journey.

Here is a link to the codebase that we are still working on and is currently under the development phase of the project.