Design Document

Team

Sudhansh Peddabomma	- 190050118
Pokala Mohith	- 190050084
Poluparthi Preetham	- 190050085
Hitesh Kumar Punna	- 190050093

3NF Schema Derivation

We started out by keeping all the attributes in a single data table and tried to figure out the functional dependencies. In doing so, we found the following functional dependencies -

- train_no → train_name, num_stations, capacity, source_id, dest_id
- path_id, train_no → station_id, expected_arrival_time, expected_departure_time
- booking_id → user_id, train_no, journey_date, start_station, end_station
- train_no, path_id, journey_date → cumulative_seats, available_seats
- station_id → station_name, zone, location
- user_id → username, age, sex, email, phone, is_admin

The above set of dependencies form the canonical cover, and we then use the 3NF decomposition algorithm to get the schemas in 3NF form. We get the set of relations seen in the schema file.

User
user_id
name
age
is_admin
phone
email
sex
password (hashed)

Station
station_id
station_name
location
zone
address

Train
train_no
train_name
capacity
num_stations
source_id
dest_id

Path
path_id
train_no
station_id
expected_arrival_time
expected_departure_time
price

Train_instance
journey_date
path_id
train_no
available_seats
cumulative_seats

Booking
booking_id
train_no
journey_date
user_id
start_station
end_station

Passenger
passenger_id
booking_id
name
age
sex
waiting_pref_no

Integrity Constraints

TABLE NAME	PRIMARY KEY	FOREIGN KEYS	NULL COLUMNS
Users	user_id	-	N/A
Train	train_no	source_id, dest_id	N/A
Station	station_id	-	Location, Zone and Address can be NULL when the data is not available

Path	path_id train_no	train_no (train) station_id(station)	Expected Arrival Time is NULL for the start station and Expected Departure Time is NULL for the end station.
Booking	booking_id	user_id (Users) start_station (Path) end_station (Path)	N/A

Train_instance	train_no
	path_id
	journey_date
		train_no(train)
		path_id(path)
			N/A

Passenger	passenger_id booking_id	booking_id(booking)	N/A

Views

Views can be created for various statistics over the data. For example, we can view all the trains in a given zone using the following view.

```
CREATE VIEW user_bookings AS
  SELECT zone, count(train_no)
  FROM  (SELECT *
  FROM  PATH
  JOIN TRAIN on TRAIN.train_no = PATH.train_no
  JOIN STATION on STATION.station_id = PATH.station_id)  as temp_zones
  GROUP BY zone
```

Queries and Data

All the update, create and delete (transactions) are written in the queries file and insert files, which are present in this folder. We have taken the data from this repository.

We have tested the queries with the data that we have loaded, and they work as expected. We plan to test the synthetic data for Users, Bookings, and Passengers more extensively during development.

We have written a script that generates synthetic data for users and bookings.

The queries for various use cases are listed below.

```
-- 1) Sign up for RailIndia -
INSERT INTO USER (user_name, email, number, log_password, age)
VALUES ($user_name, $email, $number, $log_password, $age);

-- 2) Logging into RailIndia -
SELECT is_admin FROM USER WHERE user_name = $user_name AND password =
$log_password;

-- 3) Book Ticket -
INSERT INTO BOOKING
(booking_time,train_no,journey_date,start_station,end_station,user_id)
VALUES
($booking_time,$train_no,$journey_date,$start_station,$end_station,$user
_id)

-- 4) View Available trains -
SELECT train_no FROM
(SELECT * from TRAIN_DATES
JOIN PATH on PATH.train_no=TRAIN_DATES.train_no and PATH.path_id =
TRAIN_DATES.path_id
WHERE station_id= $from_station ) as from_trains
JOIN (SELECT * from PATH WHERE station_id= $to_station ) as to_trains
on from_trains.train_no = to_trains.train_no and from_trains.date =
to_trains.date
WHERE from_trains.path_idx < to_trains.path_idx

-- 5) View Stations -
SELECT train_no FROM PATH WHERE station_id = $station_id

-- 6) Add a new train -
INSERT INTO TRAIN
(train_no,train_name,capcity,num_stations,source_id,dest_id)
VALUE
($train_no, $train_name, $capcity, $num_stations, $source_id, $dest_id)
WHERE EXISTS
(SELECT * FROM USER WHERE user_id = $user_id and is_admin =1 )

-- 7) View Train Schedule -
SELECT train_no, path_id, STATION.station_name
FROM PATH
JOIN STATION on STATION.station_id = PATH.train_id

-- 8) Find Nearest Railway Station -
SELECT *
FROM (SELECT *,
 RANK() OVER (
ORDER BY distance(location, $location)  ASC
) location_rank
FROM STATION) as Ranked_stations
```

```
WHERE location_rank == 1

-- 9) View ticket -
SELECT *
FROM BOOKING
WHERE user_id = $user_id

-- 10) Statistics -
Number of trains in each zone:-
  SELECT zone, count(train_no)
     FROM  (SELECT *
     FROM  PATH
     JOIN TRAIN on TRAIN.train_no = PATH.train_no
     JOIN STATION on STATION.station_id = PATH.station_id)  as
temp_zones
     GROUP BY zone
Number of trains in each state:-
  SELECT state, count(train_no)
     FROM  (SELECT *
     FROM  PATH
     JOIN TRAIN on TRAIN.train_no = PATH.train_no
     JOIN STATION on STATION.station_id = PATH.station_id)  as
temp_zones
     GROUP BY state

-- 11) Add a Station -
INSERT INTO STATION ( station_id, name, latitude, longitude, city,
station, zone)
VALUES ( $station_id, $name, $latitude, $longitude, $city, $station,
$zone);

-- 12) Update Waiting List -
UPDATE PASSENGER
SET waiting_pref_no=min(0,-(SELECT available_seats
FROM BOOKING
JOIN TRAIN on TRAIN.train_no = BOOKING.train_no
WHERE BOOKING.passenger_id = $passenger_id ))
WHERE Passenger_id=$passenger_id;

-- 13) Cancel Ticket -
DELETE FROM BOOKING
WHERE booking_id = $booking_id

DELETE FROM PASSENGER
WHERE booking_id = $booking_id
UPDATE PASSENGER
SET waiting_pref_no = waiting_pref_no-1
WHERE waiting_pref_no >0

-- 14) Waiting list size -
min(0,-(SELECT available_seats
FROM BOOKING
JOIN TRAIN on TRAIN.train_no = BOOKING.train_no
```

```
WHERE BOOKING.passenger_id = $passenger_id ))
Waiting list Position:-
SELECT waiting_pref_no
     FROM PASSENGER
     WHERE passenger_id = $passenger_id and booking_id = $booking_id
```

Indices

We have noticed a few places where indices will prove to be useful.

- We can set **station_id** as an index in the **Path table**, as we need to query the list of trains from a given station to another station frequently.
- **Train_no** and **Journey_date** can be added as indices in the **Booking** table as we need to query passengers for updating the waiting list of a train on a particular day.
- **Booking_id** can be made as an index in the Passenger query.
- **User_id** can be made as an index in the **Booking** table for fetching all the bookings for a particular user.

Technologies and Sketches

We plan to use a NodeJS server for the backend, PgAdmin4 for database management, and ReactJS for the frontend. We are using Python for generating data loading scripts and data preprocessing. This software will be enough for almost the entire application.

The sketches for user forms are available in the requirements and analysis document.

Business Logic Controller

We explain the logic on each page/use case drawn in the above forms.The queries can be found in github

Login Page

- **Frontend** - Does the form validation like username and password are required, the username should not contain characters outside the allowed set, etc.
- **Backend** - Calls the query script for the database. On a successful login, the user is redirected to the homepage.
- **Database** - We use **logging into database** query on database

Booking Page

- **Frontend** - Does the form validation like start and end station needed for the list of trains to be displayed.

- **Backend** - It receives the start and end stations, along with the date which is required for the journey date in the Booking. We check if there are sufficient seats for the given path. If not, the passengers are voluntarily added to the waiting list.
  If seats are available, we update the seats across all stations in the path. If the passenger is added to the waiting list, we update their waiting_pref_no, and also reduce the available_seats by 1. So if the available_seats is -6, then there are 6 people in the waiting queue.
- **Database -** We use the **booking ticket** query on the database.

Signup Page

- **Frontend** - Does the form validation like username and password are required, username should not contain characters outside the allowed set, etc. In the case of confirm password, both passwords must match. The form will only allow passwords with more than 8 characters, with atleast 1 special character and other such rules.
- **Backend** - Checks if an entry already exists in the users table with the given username or email. In that case, we send the appropriate response to the frontend.
- **Database -** We use the **signup for rail india** query on database

View trains between stations between A and B

- **Frontend** - Does the form validation like A and B are string codes. This usecase will be visible on the train booking page, where the user is shown all the trains for the query stations.
- **Backend -** Checks all available paths between A and B. This is done via a Python script that runs an online query to find the trains between the given two stations.
- **Database -** Uses **view_available_train** query

View trains passing through station

- **Frontend** - Checks if station code or station name is string and not null. This usecase will be visible on the Station info page for a particular station.
- **Backend -** Checks all trains going from given station (path table).
- **Database -** Uses **view_station** query

Add a train Page

- **Frontend** - Does the form validation like time format,station names etc and not null
- **Backend -** Checks if data is valid, checks permissions and updates the database(train, path tables)
- **Database -** Uses **add_train** query

Train Page

- **Frontend** - Does the form validation like train name or number is not null. This usecase will be visible on the Train information page for a particular train.
- **Backend -** Checks path table and extract data corresponding to given train_no. Other information regarding the train like the name and capacity will be visible on the page.

- **Database -** Uses **view_train_schedule** query.

Find nearest railway station

- **Frontend** - Does the form validation like latitude and longitude values are correct and not null if entered manually . It can also fetch current location automatically
- **Backend -** Computes nearest station in Station table
- **Database -** Uses **find_nearest_railway_station** query

View ticket

- **Frontend** - Checks if ticket number(PNR) is not null
- **Backend -** Checks all bookings entries
- **Database -** Uses **view_ticket** query

Statistics

- **Frontend** - Displays info from backend
- **Backend -** checks various tables and extracts statistical information like number of trains per station /per state etc
- **Database -** uses **statistics** query

Add station

- **Frontend** - Does the form validation like latitude and longitude are valid , station_id,station_name are not null
- **Backend -** Checks if that station already exists . If not then station table is updated
- **Database -** uses **add_station** query

Cancel ticket

- **Frontend** - Does the form validation like checks if PNR is valid
- **Backend -** Removes booking entry , Update available seats at given path index and waiting list count
- **Database -** Uses **cancel_ticket** query

Waiting list size

- **Frontend** - Checks if
- **Backend -** Checks all trains going from given station
- **Database -** Uses **waiting_list_size** query

Waiting list position

- **Frontend** - Does the form validation like time format,station names etc and not null
- **Backend -** Checks if data is valid , checks permissions and updates the database
- **Database -** uses **Waiting list position** query

Miscellaneous

- We will store password hashed using a library in Python/Javascript in the Users table.
- We have updated the ER diagram due to some issues we have found in our implementation.

Note. We have decided to hold the plans for composite trains and live tracking until we finish the main booking system. We have come across various challenges in a simple booking system website, and we have to implement various scripts to handle these cases. For example, we need to track the availability of seats across all stations in the path of a particular train. All of this needs to be carefully analyzed. Therefore, we will implement live tracking and composite trains after we are done with this.