Design Document

Team

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

3NF Schema Derivation

We started 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 → name, 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. To test the queries, we have generated synthetic data for Train No. 12797 on the date 22/04/05.

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 (name, email, number, log_password, age,sex)
VALUES ($name, $email, $number, $log_password, $age, $sex);

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

-- 3) Book Ticket -
INSERT INTO BOOKING (train_no, journey_date, start_station, end_station,
user_id)
VALUES ($train_no, $journey_date, $start_station, $end_station,
$user_id) returning booking_id;


UPDATE TRAIN_INSTANCE
SET
available_seats = available_seats - $num_seats
WHERE
train_no=$train_no and journey_date = $date and path_id>=$start_index
and path_id <= $end_index
AND EXISTS
(SELECT * FROM USER WHERE user_id = $user_id and is_admin =True );

--Find seat availability and add wc,seat_no accordingly to passengers
entry,the first insert gives booking id
INSERT INTO Passenger(booking_id, name, seat_no, age, sex,
waiting_pref_no)
VALUES ( $bid, $name, $seatno, $age, $sex, $wc_no);

-- 4) View Available trains -
--Multipath
with f(train1,train2,d1,d2,total_dist,halt_station, deptime, arrtime) as
(
select  B.train_no,C.train_no
,B.Distance_from_source-A.Distance_from_source
,D.Distance_from_source-C.Distance_from_source,B.Distance_from_source-A.
Distance_from_source +D.Distance_from_source-C.Distance_from_source as
l,B.station_id,C.expected_departure_time,B.expected_arrival_time,
EXTRACT(EPOCH FROM (C.expected_departure_time -
B.expected_arrival_time)) AS difference
from paths as B, paths as A,paths as C,paths as D
where
B.train_no = A.train_no
and
A.path_id<B.path_id
and
D.path_id>C.path_id
and
```

```
D.train_no = C.train_no
and
C.station_id=B.station_id
and
A.station_id=$1 and D.station_id=$2
and
C.train_no!=A.train_no
and
C.expected_departure_time > B.expected_arrival_time
and
EXTRACT(EPOCH FROM (C.expected_departure_time -
B.expected_arrival_time)) < 1200
and
EXTRACT(EPOCH FROM (C.expected_departure_time -
B.expected_arrival_time)) > 300
)
select * from f
where
total_dist < (select min(total_dist) from f)*1.1
order by total_dist desc limit 3

- 5) single train info
SELECT
A.train_no
,Train_name,source_id,dest_id,B.distance_from_source-A.distance_from_sou
rce as dist,B.expected_arrival_time,A.expected_departure_time
FROM
paths as A,train as C,
paths as B where A.train_no=B.train_no and A.station_id = $1 and
B.station_id = $2
and A.train_no = C.train_no

-- 6) View Trains passing through a Station -
SELECT train_no FROM PATHs WHERE station_id = $station_id

-- 7) 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 = True);

- 8) for each train in the path
INSERT INTO
paths(path_id,train_no,station_id,distance_from_source,price,expected_ar
rival_time,expected_departure_time)
 VALUES($path_id,$train_no,$sc,$dist,$price_from_source,$eat,$edt);


-- 9) View Train Schedule -
SELECT train_no, path_id, STATION.station_name, expected_arrival_time,
expected_departure_time, distance_from_source
```

```
FROM PATHs
JOIN STATION on STATION.station_id = PATHs.station_id
and train_no=$train
order by path_id asc

-- 10) 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

-- 11) View ticket -
SELECT *
FROM BOOKING
WHERE booking_id = $PNR

-- 12) Statistics -
Number of trains in each zone:-
 SELECT zone, count(train_no)
     FROM
(SELECT TRAIN.train_no as train_no,zone
     FROM  Paths
     JOIN TRAIN on TRAIN.train_no = Paths.train_no
     JOIN STATION on STATION.station_id = Paths.station_id) as temp
        GROUP BY zone

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

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

-- 14) Cancel Ticket -
update Train_instance
set
available_seats = available_seats - (select count(*) from passenger
where booking_id=$bid);

DELETE FROM passenger where booking_id=$bid;
DELETE FROM booking where booking_id=$bid;
with A(pid,r,t) as (
      select passenger_id,rank() over (partition by
```

```
journey_date,train_no order by waiting_pref_no asc)
      ,booking.train_no from booking , passenger
where
booking.booking_id=passenger.booking_id
and  booking.start_station >= $s and booking.end_station<=$e and
booking.train_no=$t
)
update passenger
set seat_no = -A.r + (select capacity from train where
train.train_no=$t),
waiting_pref_no  = A.r - (select capacity from train where
train.train_no=$t)

FROM booking,A
where
A.pid=passenger.passenger_id;

select * from passenger


-- 15) Release Tickets
INSERT into
Train_instance(
      journey_date,
      available_seats,
      cumulative_seats,
      path_id,
train_no)
SELECT
      $date,
      $num_seats,
      $num_seats,
      path_id,
      $train_no
FROM
paths
where train_no = $train_no

-- 16) Seat Availability
select min(available_seats)
from
Train_instance
where
train_no = $train
and
path_id >=$start_index
and
path_id <=$end_index
and
journey_date = $date
```

For example, here are the outputs for some of the queries.

```
-- 6) View Trains passing through a Station -
SELECT train.train_no, train_name
FROM PATHS JOIN Train
ON paths.train_no = Train.train_no
WHERE station_id = 'DDR'
```

	train_no [PK] integer	train_name text
1	12489	BKN DDR SF E
2	12490	DDR BIKANER
3	12901	GUJARAT MAIL
4	12902	GUJARAT MAIL
5	12904	GOLDN TEMPLE
6	12928	VADODARA EXP
7	12959	DADAR BHUJ E
8	12960	BHUJ DADAR E
9	12989	DDR AII SF E
10	12990	AII DDR SF E
11	19015	SAURASHTRA E
12	19016	SAURASHTRA E
13	19023	FZR JANATA E
14	19024	FZR BCT JANT
15	19115	DDR BHUJ EXP
16	19116	BHUJ DDR EXP
17	19215	SAURASHTRA E
18	22945	SAURASHTRA M
19	22946	SAURASHTRA M
20	22953	GUJARAT EXPR
21	22954	GUJARAT EXPR
22	59439	AHMADABAD PA
23	59441	AHMEDABAD PA
24	59442	AHMEDABAD PA

	train_no integer	path_id integer	station_name text	expected_arrival_time time without time zone	expected_departure_time time without time zone	distance_from_source integer
1	12797	1	KACHEGUDA	20:05:00	20:05:00	0
2	12797	2	UMDANAGAR	20:29:00	20:30:00	20
3	12797	3	SHADNAGAR	20:55:00	20:56:00	51
4	12797	4	JADCHERLA	21:23:00	21:25:00	87
5	12797	5	MAHBUBNAGAR	21:42:00	21:44:00	105
6	12797	6	WANPARTI ROAD	22:21:00	22:22:00	158
7	12797	7	GADWAL	22:44:00	22:45:00	180
8	12797	8	KURNOOL TOWN	23:46:00	23:48:00	236
9	12797	9	DHONE	01:00:00	01:05:00	289
10	12797	10	TADIPATRI	02:38:00	02:40:00	392
11	12797	11	MUDDANURU	03:29:00	03:30:00	445
12	12797	12	YERRAGUNTLA	03:44:00	03:45:00	461
13	12797	13	KAMALAPURAM	03:59:00	04:00:00	476
14	12797	14	CUDDAPAH	04:28:00	04:30:00	500
15	12797	15	NANDALUR	05:04:00	05:05:00	540
16	12797	16	RAZAMPETA	05:19:00	05:20:00	551
17	12797	17	OBULAVARIPALLI	05:49:00	05:50:00	571
18	12797	18	KODURU	06:09:00	06:10:00	584
19	12797	19	RENIGUNTA JN	06:48:00	06:50:00	625
20	12797	20	TIRUPATI MAIN	07:15:00	07:20:00	635
21	12797	21	PAKALA JN	08:09:00	08:10:00	677
22	12797	22	CHITTOOR	08:55:00	08:55:00	707

The left image gives the list of trains passing through Dadar station. The right image gives the train schedule for the train KCG-CTO VENK having the train number 12797.

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.
- **Station Name** can be used as an index to find the corresponding Station Code for fetching trains between two stations.

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.

Login Page

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

Release tickets

- **Frontend** - Does the form validation like the valid date (date in future), valid train number
- **Backend** - Checks for permissions(if done by admin). And update train instance entry.
- **Database** - We use **release_ticket query** on database

Available seats

- **Frontend** - Does the form validation like the valid date(date in future), valid train number, valid start station, and end station
- **Backend** - Checks for seat availability across all stations from start to end and returns the minimum
- **Database** - We use **seat_availability 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 name and password are required, the name 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 name or email. In that case, we send the appropriate response to the frontend.
- **Database -** We use the **signup for rail India** query on the 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 use case 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 the 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 the number of trains per station /per state etc
- **Database -** uses **statistics** query

Add station

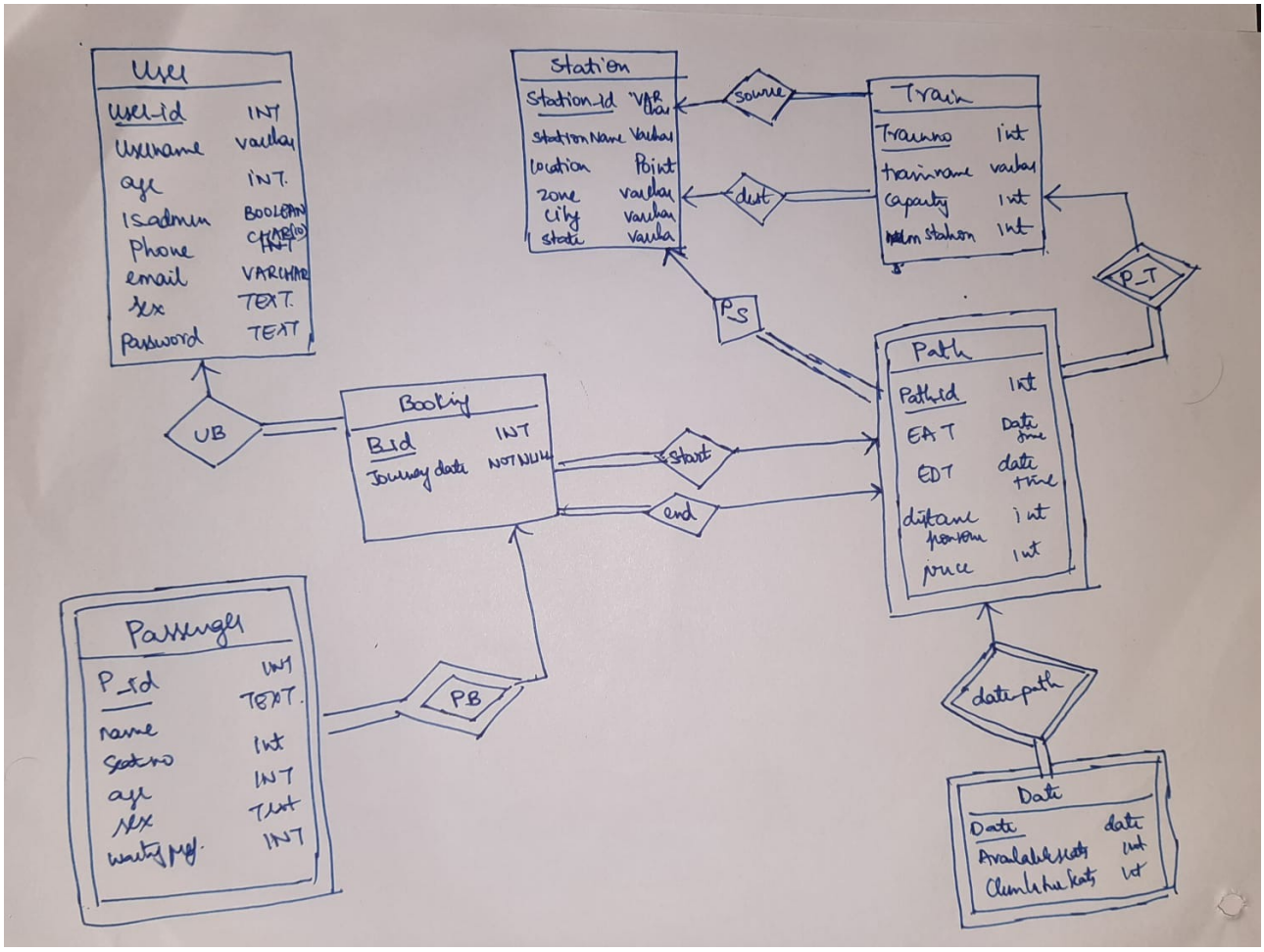
- **Frontend** - Does the form validation like latitude and longitude are valid, station_id,station_name are not null
- **Backend -** Checks if that station already exists. If not then the station table is updated
- **Database -** uses **add_station** query

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

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 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 after we are done with this.