DBMS PROJECT

SUBMITTED TO : Ms. Simran

Railway management system

Submitted by:

1)	Hardik Verma	102283044
2)	Namit Nayyar	102103831
3)	Komal	102283043
4)	Bhaavya	102103783

INDEX: **Pg NO.**

Requirement Analysis

Requirement analysis is a crucial step in the development of any software project, including a railway management database. The following are some of the key requirements that should be considered:

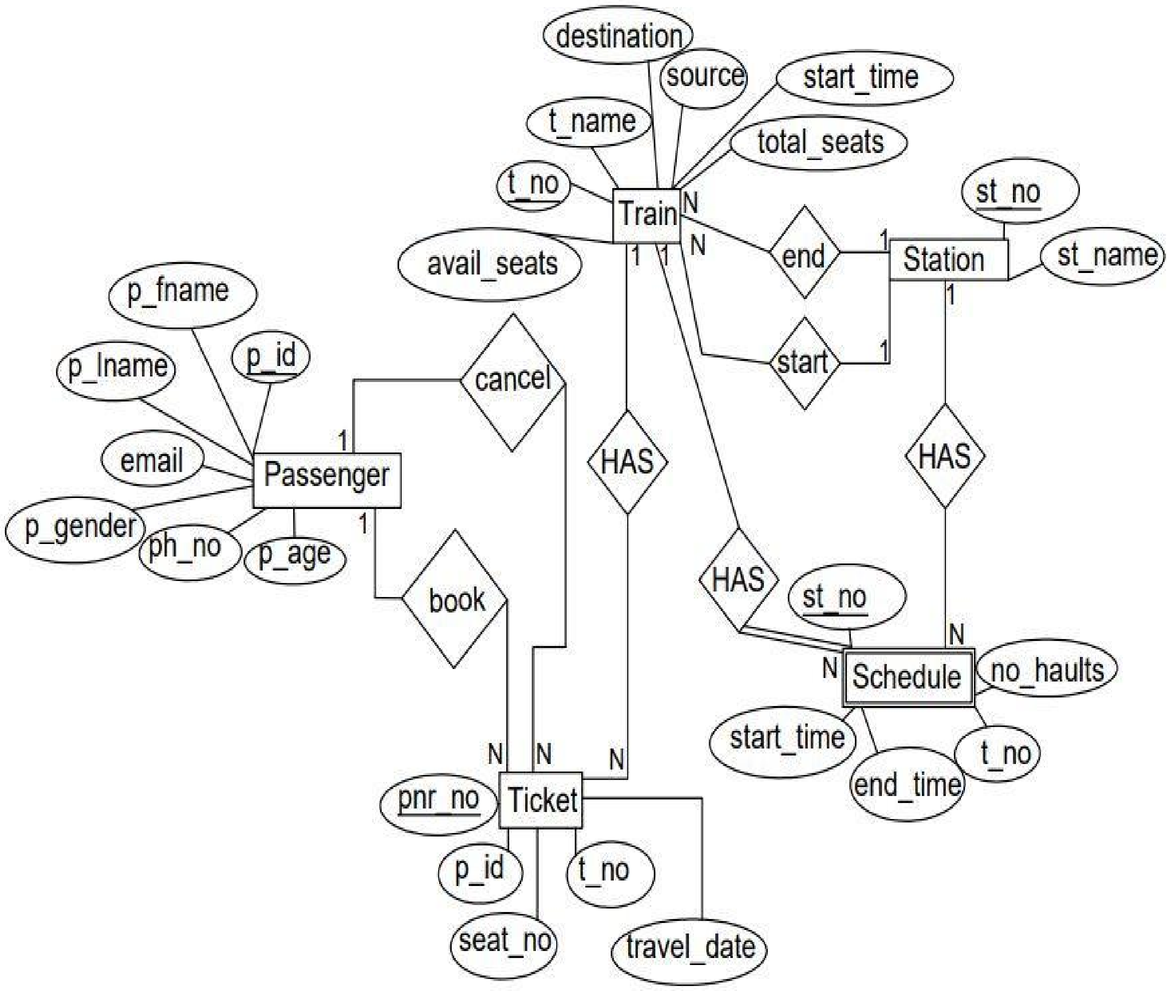
Functional requirements: The railway management database should be able to perform the following functions:

1) Keep track of train schedules, including arrival and departure times, and the routes taken by each train.
2) Manage ticket sales and reservations for each train and route.
3) Record passenger information, including names, ages, and contact information.
4) Track payment information for each ticket sold, including payment method and amount.
5) Maintain an inventory of available seats and compartments on each train.

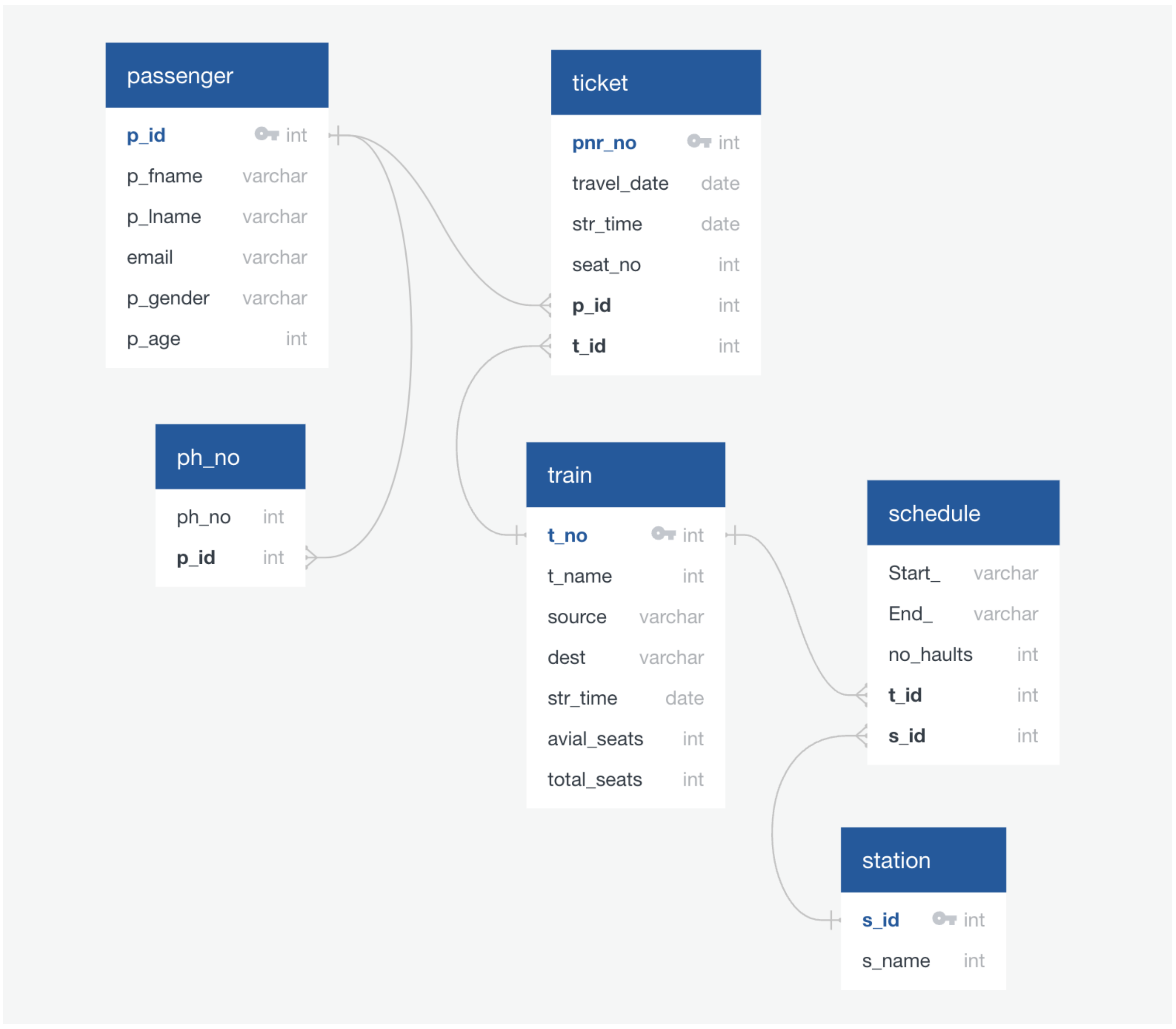
Non-functional requirements: The railway management database should meet the following non-functional requirements:

1) Security: The database should be secure and protected from unauthorized access.
2) Scalability: The database should be able to handle large amounts of data and be scalable to accommodate future growth.
3) Reliability: The database should be reliable and able to handle high levels of traffic without downtime or data loss.
4) Performance: The database should be optimized for fast access and retrieval of data.
5) User-friendliness: The database should be easy to use and navigate for both railway staff and passengers.

ER DIAGRAM:


destination
source
start_time
t_name
total_seats
t_no
Train
avail_seats
N
1
1
N
end
1
Station
st_no
st_name
start
1
1
p_fname
p_id
p_lname
cancel
HAS
HAS
email
Passenger
1
p_gender
ph_no
p_age
1
book
HAS
st_no
N
N
Schedule
no_haults
start_time
end_time
t_no
N
N
N
pnr_no
Ticket
p_id
t_no
seat_no
travel_date

ER TO TABLE:


passenger
p_id int
p_fname varchar
p_lname varchar
email varchar
p_gender varchar
p_age int
ph_no
ph_no int
p_id int
ticket
pnr_no int
travel_date date
str_time date
seat_no int
p_id int
t_id int
train
t_no int
t_name int
source varchar
dest varchar
str_time date
avial_seats int
total_seats int
schedule
Start_ varchar
End_ varchar
no_haults int
t_id int
s_id int
station
s_id int
s_name int

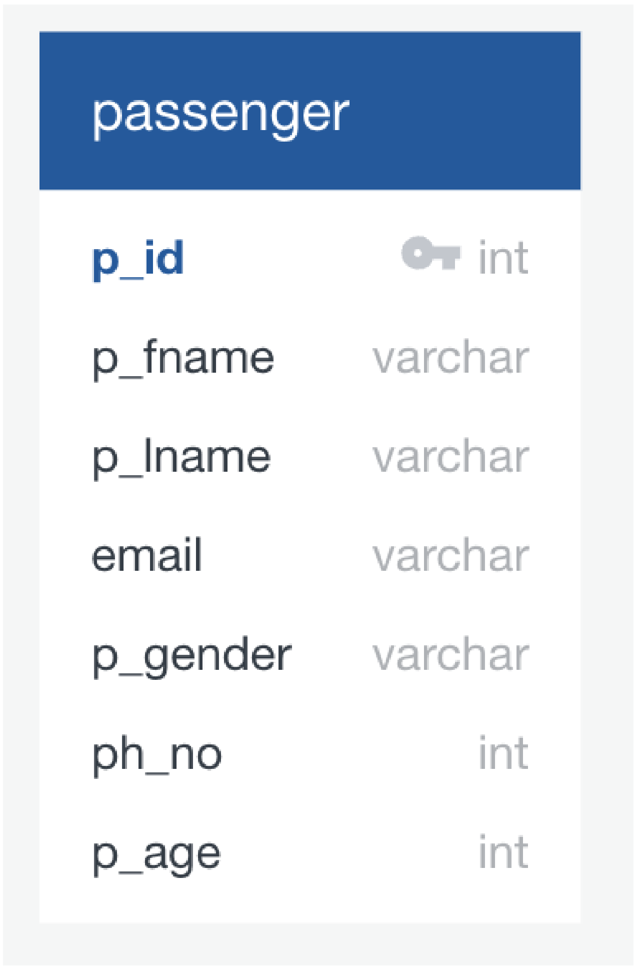
NORMALISATION:

FIRST NORMAL FORM:

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Since a passenger could have multiple phone numbers ,it would violate the 1NF rules. Hence we have created a separate table called contact to handle this.

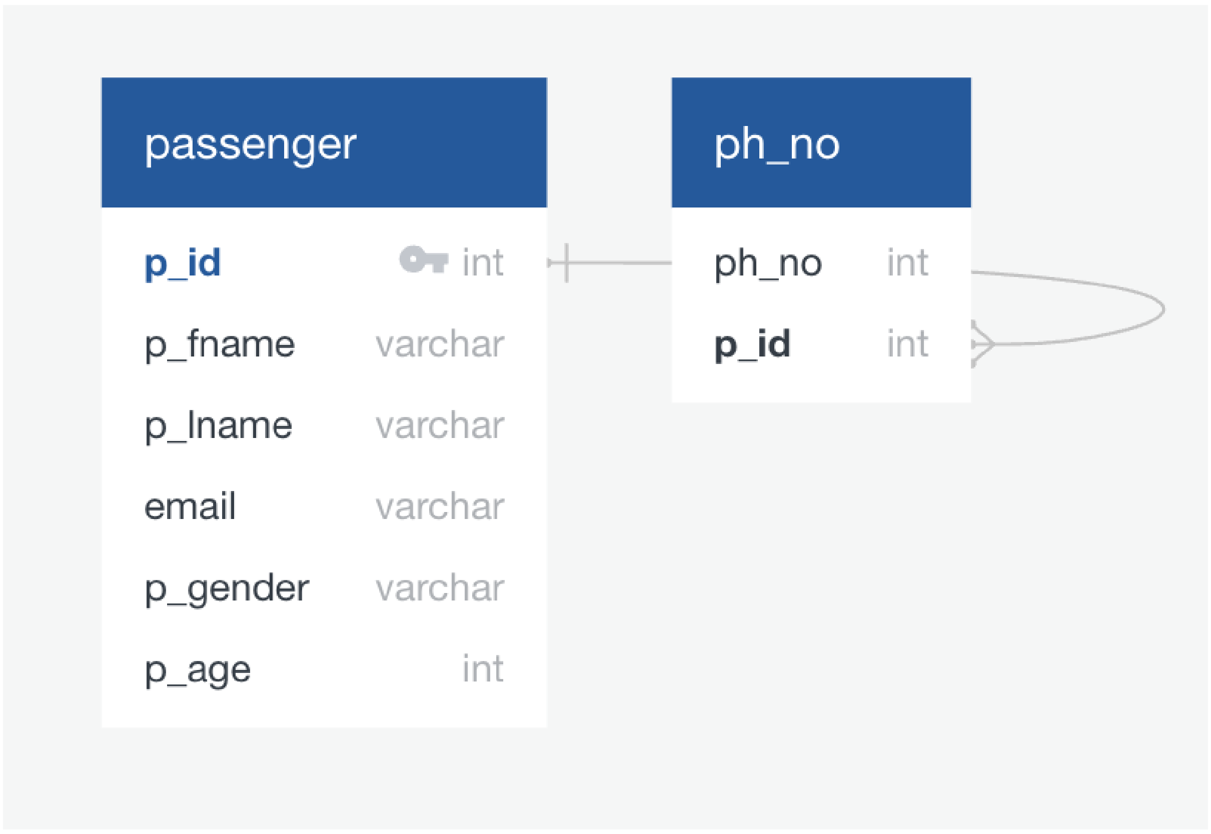
Before Normalisation:



passenger	
p_id	int
p_fname	varchar
p_lname	varchar
email	varchar
p_gender	varchar
ph_no	int
p_age	int

After normalisation:

```
 CREATE TABLE phone_numbers (
  phone_number VARCHAR2(20) NOT NULL,
  p_id INT REFERENCES passenger(p_id)
);
```

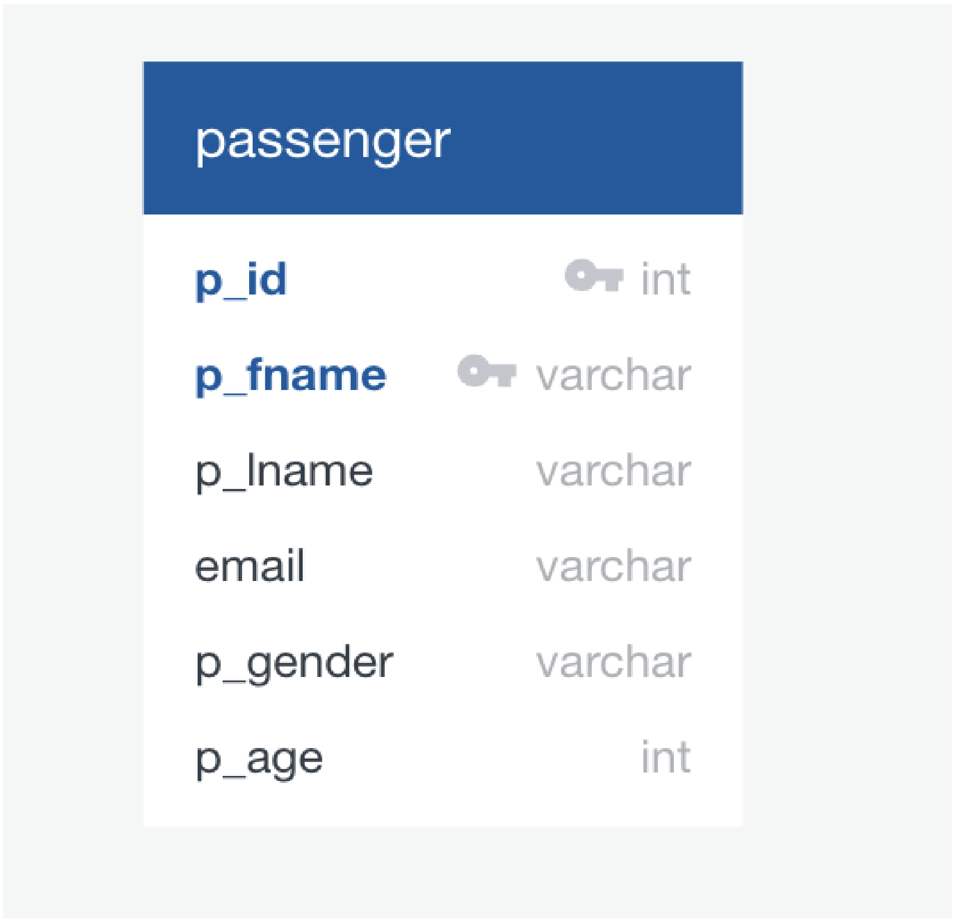


SECOND NORMAL FORM:

A table is said to be in 2NF if both the following conditions hold:
-Table is in 1NF (First normal form)
-No non-prime attribute is dependent on the proper subset of any candidate key of table.

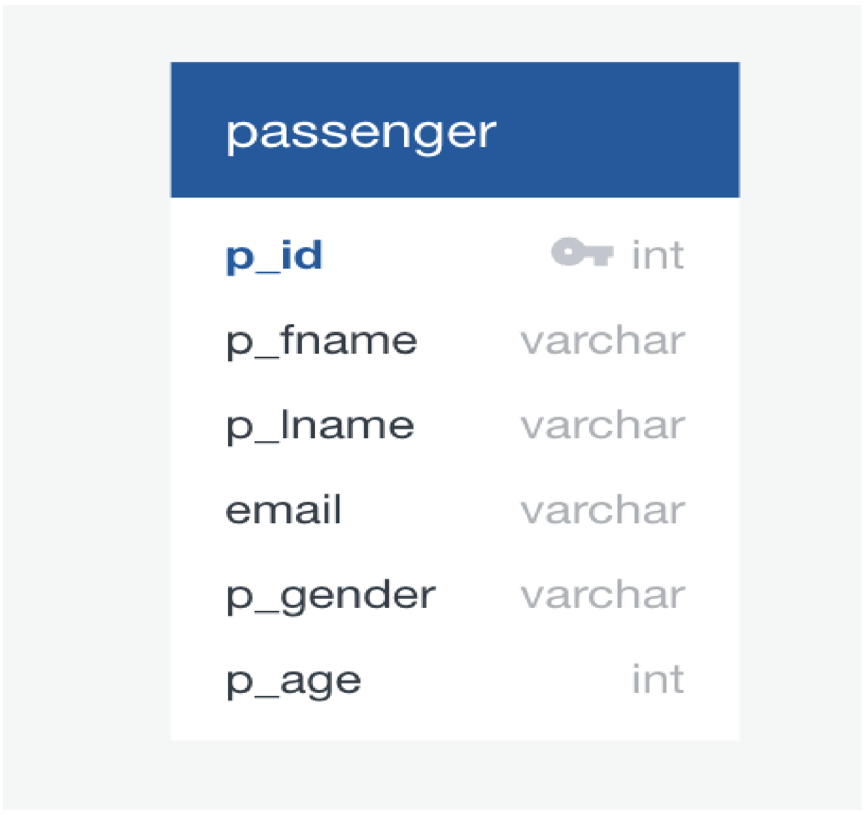
If in Passenger table we consider ticket_no and first_name as the candidate key,then date_of_birth would depend only on the name and it would violate the 2NF.

Before normalisation :



passenger	
p_id	🔑 int
p_fname	🔑 varchar
p_lname	varchar
email	varchar
p_gender	varchar
p_age	int

After normalisation:



passenger	
p_id	🔑 int
p_fname	varchar
p_lname	varchar
email	varchar
p_gender	varchar
p_age	int

THIRD NORMAL FORM:

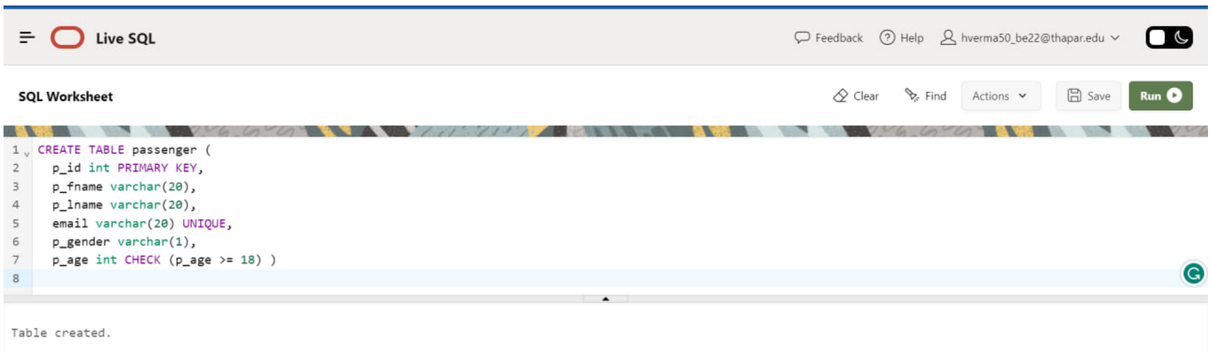
A table design is said to be in 3NF if both the following conditions hold:

-Table must be in 2NF
-Transitive functional dependency of non-prime attribute on any super key should be removed.

Our schema follows the above rules and hence is in 3NF.

--Passenger Table

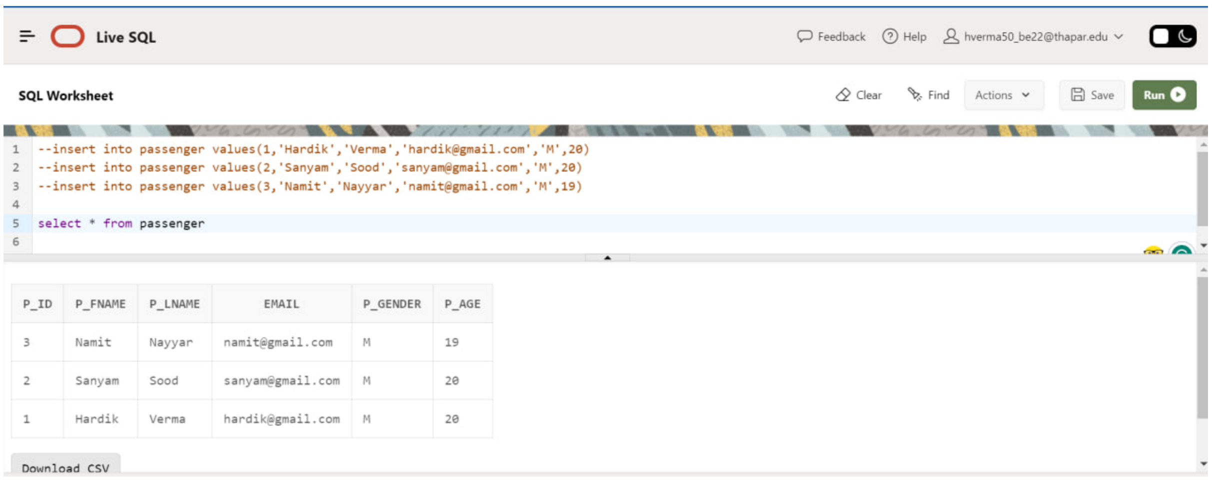
```
CREATE TABLE passenger (
  p_id int PRIMARY KEY,
  p_fname varchar(20),
  p_lname varchar(20),
  email varchar(20) UNIQUE,
  p_gender varchar(1),
  p_age int CHECK (p_age >= 18) )
```



```
CREATE TABLE passenger (
  p_id int PRIMARY KEY,
  p_fname varchar(20),
  p_lname varchar(20),
  email varchar(20) UNIQUE,
  p_gender varchar(1),
  p_age int CHECK (p_age >= 18) )
```

```
Table created.
```

```
insert into passenger
values(1,'Hardik','Verma','hardik@gmail.com','M',20)
insert into passenger
values(2,'Sanyam','Sood','sanyam@gmail.com','M',20)
insert into passenger
values(3,'Namit','Nayyar','namit@gmail.com','M',19)
```



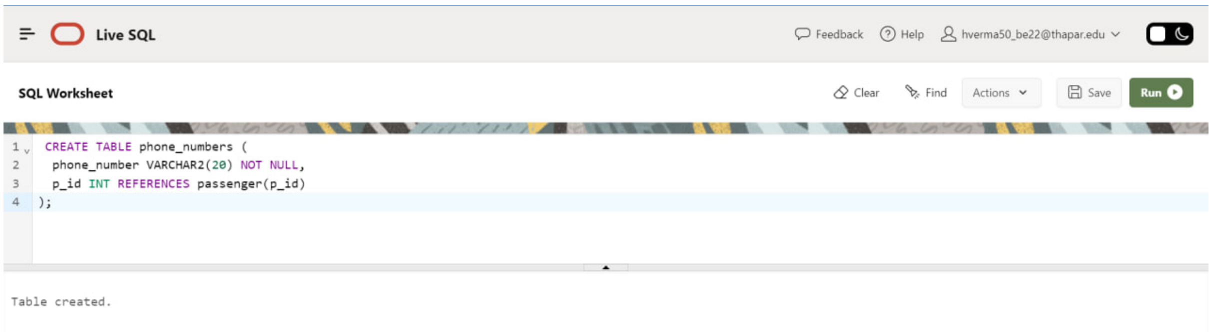
```
--insert into passenger values(1,'Hardik','Verma','hardik@gmail.com','M',20)
--insert into passenger values(2,'Sanyam','Sood','sanyam@gmail.com','M',20)
--insert into passenger values(3,'Namit','Nayyar','namit@gmail.com','M',19)

select * from passenger
```

P_ID	P_FNAME	P_LNAME	EMAIL	P_GENDER	P_AGE
3	Namit	Nayyar	namit@gmail.com	M	19
2	Sanyam	Sood	sanyam@gmail.com	M	20
1	Hardik	Verma	hardik@gmail.com	M	20

--Phone Numbers Table (multivalued):

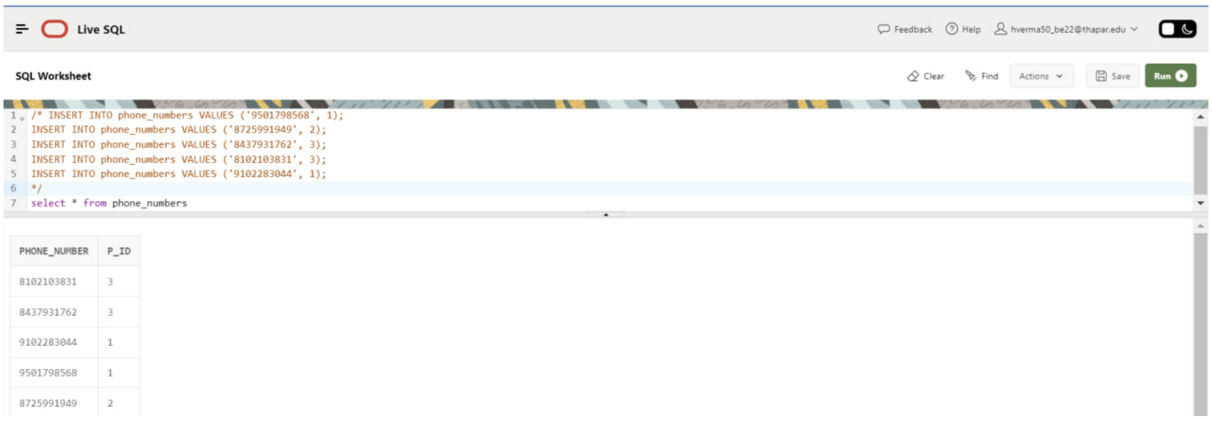
```
 CREATE TABLE phone_numbers (
  phone_number VARCHAR2(20) NOT NULL,
  p_id INT REFERENCES passenger(p_id)
);
```



```
CREATE TABLE phone_numbers (
  phone_number VARCHAR2(20) NOT NULL,
  p_id INT REFERENCES passenger(p_id)
);
```

```
Table created.
```

```
INSERT INTO phone_numbers VALUES ('9501798568', 1);
INSERT INTO phone_numbers VALUES ('8725991949', 2);
INSERT INTO phone_numbers VALUES ('8437931762', 3);
INSERT INTO phone_numbers VALUES ('8102103831', 3);
INSERT INTO phone_numbers VALUES ('9102283044', 1);
```

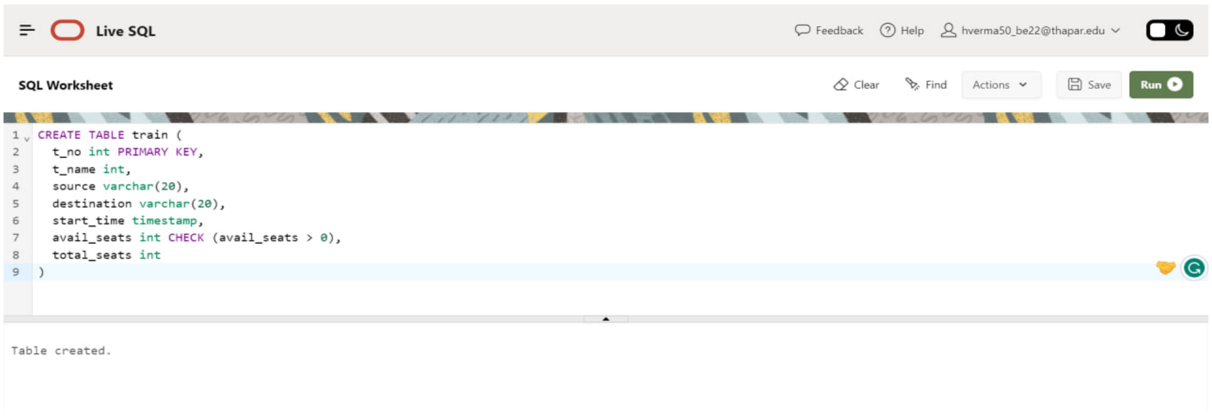


```
/* INSERT INTO phone_numbers VALUES ('9501798568', 1);
INSERT INTO phone_numbers VALUES ('8725991949', 2);
INSERT INTO phone_numbers VALUES ('8437931762', 3);
INSERT INTO phone_numbers VALUES ('8102103831', 3);
INSERT INTO phone_numbers VALUES ('9102283044', 1);
*/
select * from phone_numbers
```

PHONE_NUMBER	P_ID
8102103831	3
8437931762	3
9102283044	1
9501798568	1
8725991949	2

--Train Table:

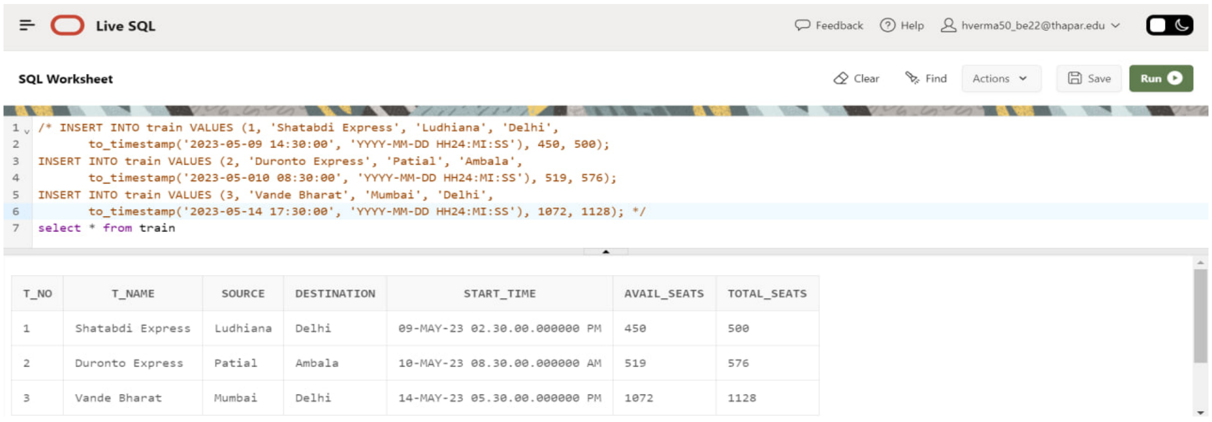
```
CREATE TABLE train (
  t_no int PRIMARY KEY, t_name varchar(50),
  source varchar(20), destination varchar(20),
  start_time timestamp, avail_seats int CHECK (avail_seats >= 0),
  total_seats int
)
```



```
CREATE TABLE train (
  t_no int PRIMARY KEY,
  t_name int,
  source varchar(20),
  destination varchar(20),
  start_time timestamp,
  avail_seats int CHECK (avail_seats > 0),
  total_seats int
)
```

```
Table created.
```

```
INSERT INTO train VALUES (1, 'Shatabdi Express', 'Ludhiana', 'Delhi',
to_timestamp('2023-05-09   14:30:00',   'YYYY-MM-DD   HH24:MI:SS'),
450, 500);
INSERT INTO train VALUES (2, 'Duronto Express', 'Patial', 'Ambala',
to_timestamp('2023-05-010  08:30:00',  'YYYY-MM-DD  HH24:MI:SS'),
519, 576);
INSERT INTO train VALUES (3, 'Vande Bharat', 'Mumbai', 'Delhi',
to_timestamp('2023-05-14   17:30:00',   'YYYY-MM-DD   HH24:MI:SS'),
1072, 1128);
```



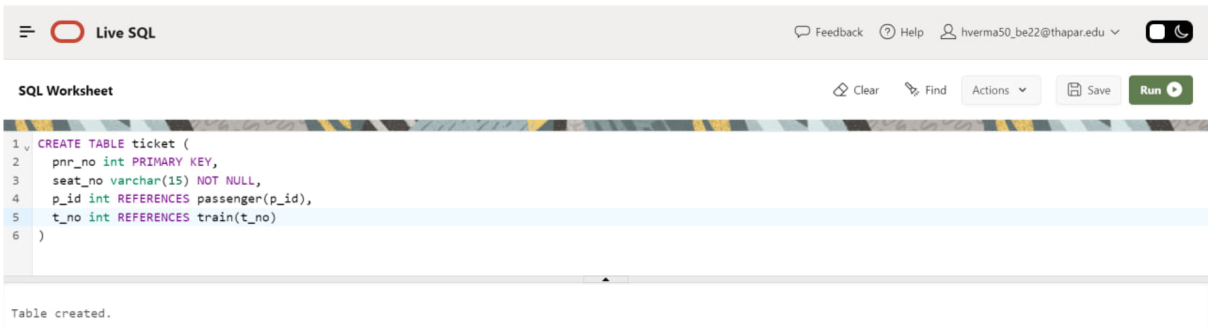
```
/* INSERT INTO train VALUES (1, 'Shatabdi Express', 'Ludhiana', 'Delhi',
       to_timestamp('2023-05-09 14:30:00', 'YYYY-MM-DD HH24:MI:SS'), 450, 500);
INSERT INTO train VALUES (2, 'Duronto Express', 'Patial', 'Ambala',
       to_timestamp('2023-05-010 08:30:00', 'YYYY-MM-DD HH24:MI:SS'), 519, 576);
INSERT INTO train VALUES (3, 'Vande Bharat', 'Mumbai', 'Delhi',
       to_timestamp('2023-05-14 17:30:00', 'YYYY-MM-DD HH24:MI:SS'), 1072, 1128); */
select * from train
```

T_NO	T_NAME	SOURCE	DESTINATION	START_TIME	AVAIL_SEATS	TOTAL_SEATS
1	Shatabdi Express	Ludhiana	Delhi	09-MAY-23 02.30.00.000000 PM	450	500
2	Duronto Express	Patial	Ambala	10-MAY-23 08.30.00.000000 AM	519	576
3	Vande Bharat	Mumbai	Delhi	14-MAY-23 05.30.00.000000 PM	1072	1128

--Ticket Table:

CREATE TABLE ticket (
  pnr_no int PRIMARY KEY, seat_no varchar(15) NOT NULL,

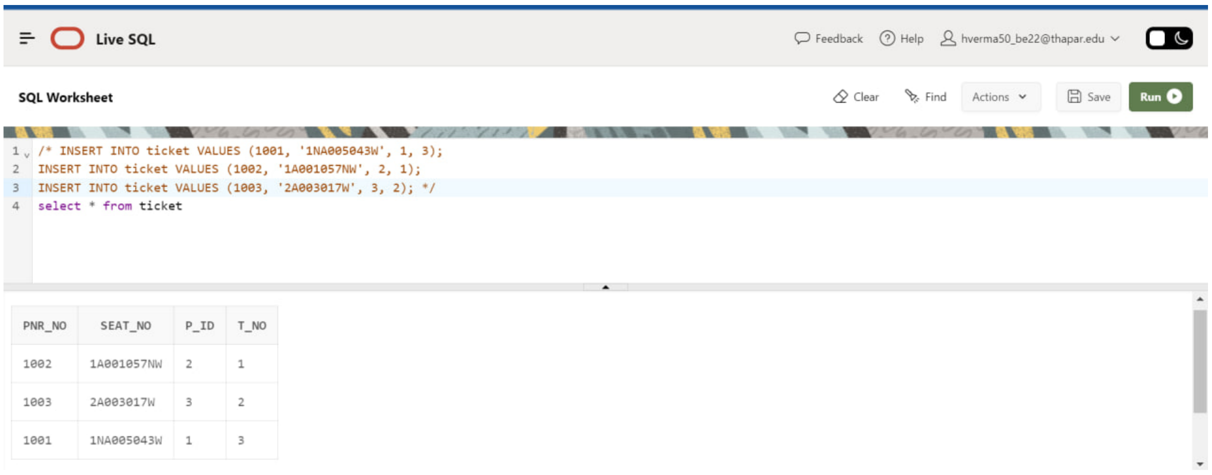
  p_id int REFERENCES passenger(p_id), t_id int REFERENCES train(t_id))



```
CREATE TABLE ticket (
  pnr_no int PRIMARY KEY,
  seat_no varchar(15) NOT NULL,
  p_id int REFERENCES passenger(p_id),
  t_no int REFERENCES train(t_no)
)
```

```
Table created.
```

INSERT INTO ticket VALUES (1001, '1NA005043W', 1, 3);
INSERT INTO ticket VALUES (1002, '1A001057NW', 2, 1);
INSERT INTO ticket VALUES (1003, '2A003017W', 3, 2);

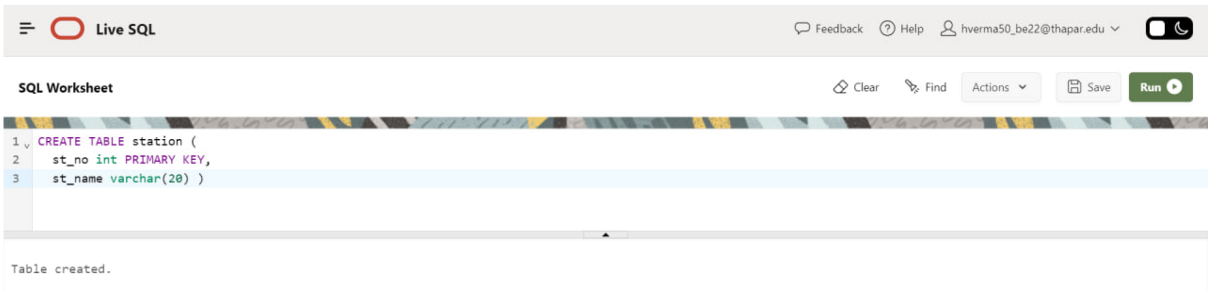


```
/* INSERT INTO ticket VALUES (1001, '1NA005043W', 1, 3);
INSERT INTO ticket VALUES (1002, '1A001057NW', 2, 1);
INSERT INTO ticket VALUES (1003, '2A003017W', 3, 2); */
select * from ticket
```

PNR_NO	SEAT_NO	P_ID	T_NO
1002	1A001057NW	2	1
1003	2A003017W	3	2
1001	1NA005043W	1	3

--Station Table

CREATE TABLE station (st_no int PRIMARY KEY, st_name varchar(20))



```
CREATE TABLE station (
  st_no int PRIMARY KEY,
  st_name varchar(20) )
```

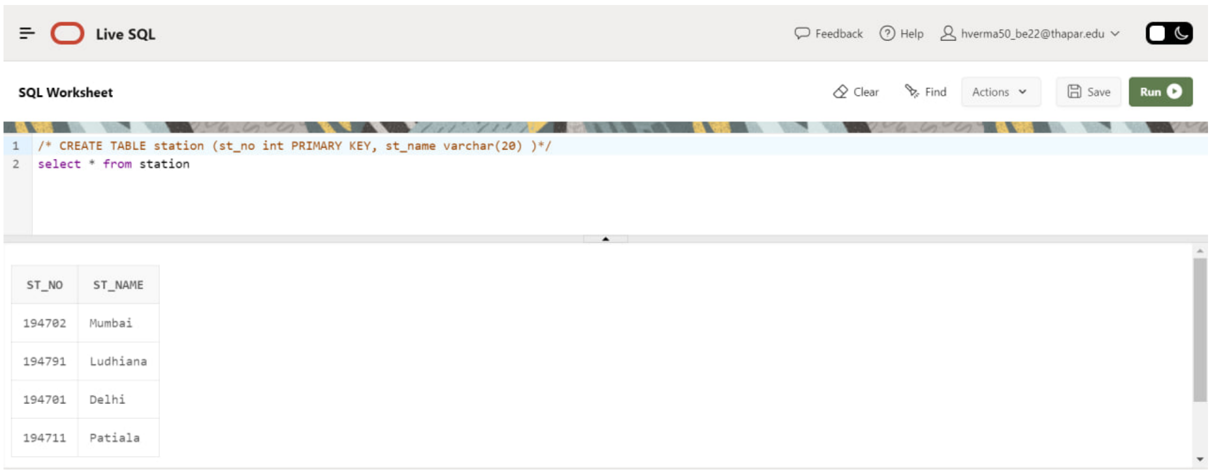
```
Table created.
```

insert into station values(194701, 'Delhi')

insert into station values(194702, 'Mumbai')

insert into station values(194791, 'Ludhiana')

insert into station values(194711, 'Patiala')

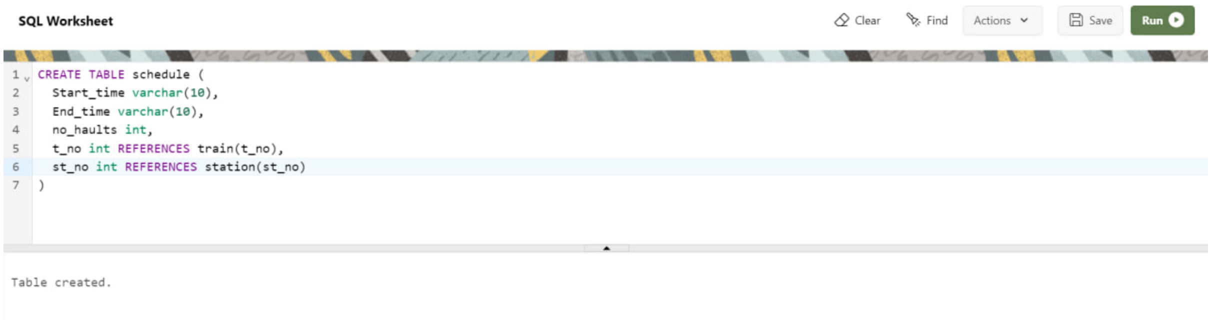


```
/* CREATE TABLE station (st_no int PRIMARY KEY, st_name varchar(20) )*/
select * from station
```

ST_NO	ST_NAME
194702	Mumbai
194791	Ludhiana
194701	Delhi
194711	Patiala

--Schedule Table:

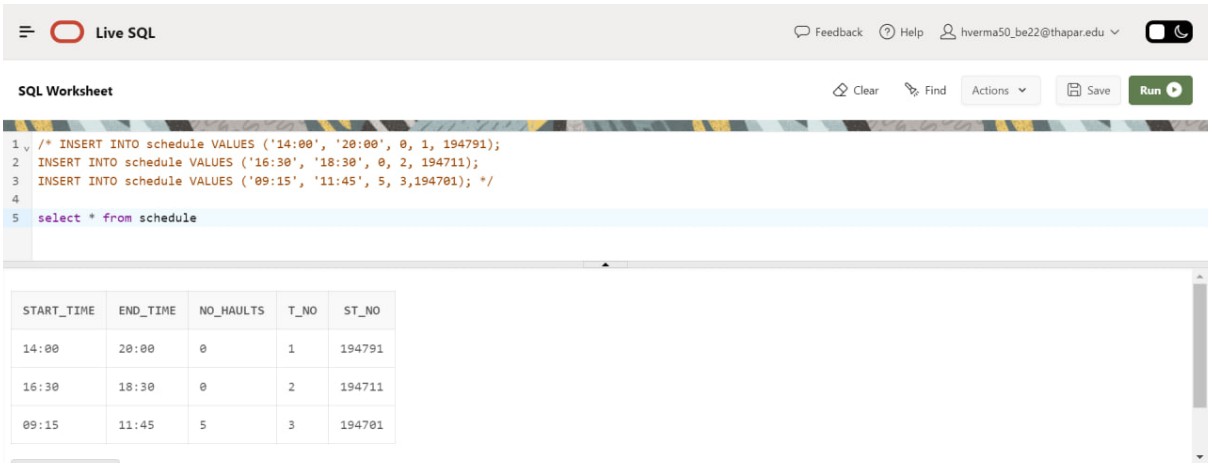
```
CREATE TABLE schedule (
  Start_time varchar(10),
  End_time varchar(10),
  no_haults int,
  t_no int REFERENCES train(t_no),
  st_no int REFERENCES station(st_no)
)
```


SQL Worksheet

```
CREATE TABLE schedule (
  Start_time varchar(10),
  End_time varchar(10),
  no_haults int,
  t_no int REFERENCES train(t_no),
  st_no int REFERENCES station(st_no)
)
```

```
Table created.
```

```
INSERT INTO schedule VALUES ('14:00', '20:00', 0, 1, 194791);
INSERT INTO schedule VALUES ('16:30', '18:30', 0, 2, 194711);
INSERT INTO schedule VALUES ('09:15', '11:45', 5, 3,194701);
```



SQL Worksheet

```
/* INSERT INTO schedule VALUES ('14:00', '20:00', 0, 1, 194791);
INSERT INTO schedule VALUES ('16:30', '18:30', 0, 2, 194711);
INSERT INTO schedule VALUES ('09:15', '11:45', 5, 3,194701); */

select * from schedule
```

START_TIME	END_TIME	NO_HAULTS	T_NO	ST_NO
14:00	20:00	0	1	194791
16:30	18:30	0	2	194711
09:15	11:45	5	3	194701

Triggers

1). Trigger to update the available seats in the Train table after a new ticket is booked:

```
CREATE TRIGGER update_avial_seats
AFTER INSERT ON ticket
FOR EACH ROW
UPDATE train
SET avial_seats = avial_seats - 1
WHERE t_no = NEW.t_id;
```

2). Trigger to prevent inserting a new Passenger if the email or phone number already exists in the database:

```
CREATE TRIGGER check_duplicate_contact_info
BEFORE INSERT ON passenger
FOR EACH ROW
BEGIN
  IF (EXISTS (SELECT 1 FROM passenger WHERE email = NEW.email OR
ph_no = NEW.ph_no)) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Email or phone number already exists';
  END IF;
END;
```

3). Trigger to prevent inserting a new Ticket if the specified seat number is already booked:

```
CREATE TRIGGER check_duplicate_seat_number
BEFORE INSERT ON ticket
FOR EACH ROW
```

```
BEGIN
  IF (EXISTS (SELECT 1 FROM ticket WHERE seat_no = NEW.seat_no
AND t_id = NEW.t_id)) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Seat already booked';
  END IF;
END;
```

4). Trigger to update the available seats in the Train table after a Ticket is canceled:

```
CREATE TRIGGER update_avial_seats_on_delete
AFTER DELETE ON ticket
FOR EACH ROW
UPDATE train
SET avial_seats = avial_seats + 1
WHERE t_no = OLD.t_id;
```

Procedures:

Procedure to get the list of Passengers with their booked Tickets for a specific Train:

```
CREATE PROCEDURE get_passengers_with_tickets(
  IN t_no int
)
BEGIN
  SELECT p.p_id, p.p_fname, p.p_lname, p.email, p.p_gender,
p.p_age, t.pnr_no, t.t_amount, t.t_date, t.str_time, t.seat_no
  FROM passenger p
  JOIN ticket t ON p.p_id = t.p_id
  WHERE t.t_id = t_no;
END;
```

Procedure to get the number of available seats for a specific Train:

```
CREATE PROCEDURE get_available_seats(
  IN t_no int,
  OUT avial_seats int
)
BEGIN
  SELECT avial_seats INTO avial_seats FROM train WHERE t_no =
t_no;
END;
```

CONCLUSION:

The railway management database schema presented in this project provides a solid foundation for managing data related to a railway system. It includes tables for passengers, tickets, trains, schedules, and stations, and is designed to support the core functions of a railway system, such as managing passengers, tracking train schedules, and selling tickets.

While the schema is well-structured and normalized, there may be additional tables or columns that could be added to support more advanced features, such as managing train maintenance schedules or tracking luggage. Additionally, the schema could benefit from additional constraints and triggers to ensure data integrity and automate certain tasks.

Overall, this railway management database schema provides a starting point for building a comprehensive railway management system. With additional features and functionality added, it could be a powerful tool for managing railway operations and improving the passenger experience.

References:

1. https://www.ques10.com/p/9477/draw-e-r-diagram-for-online-ticket-railway-reser-1/
2. https://app.quickdatabasediagrams.com/#/d/YLY8XE