DBMS PROJECT PRESENTATION

Group-5 (TA- Rahul Gupta)

Mudit Gupta (2020315)
Srijan Arora (2020342)
Srishti Jain (2020543)
Siya Garg (2020577)

SCOPE OF PROJECT

- We have tried to implement the Railway Reservation System like IRCTC. We have included a **train**, **bus**, and **flight** reservation and booking system along with a **payment portal**.

- We have assumed that all the transports run every day and reach the destination **within 24 hours** for the simplicity of our database.

- Our reservation system for all transports works on the following basis: Each vehicle has **multiple stations** associated with it where it stops, and travelers can board and un-board it there. The user will input the **source** and **destination** stations and then choose the train from the displayed options to cater to this.

- We have an **admin** who has access to the sql database.

STAKEHOLDERS

01. User/Traveler/Client

Users can log in to the website using their user id and password and book tickets for buses, flights, and trains.

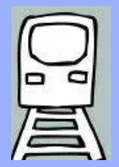
02. Admin

Admin has access to SQL database and hence can make some changes to the database and can view tables which a user/client cannot view.

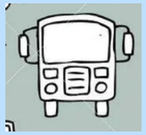
03. Indian Railway Management

They will have access as an admin to view the number of seats booked, seats vacant, user information, etc. Simply put, they can access all the data present in the database.

TERNARY RELATIONSHIPS



Train_booking between the user, train, station, and payment



Bus_booking between the user, bus, bus_stop, and payment

Flight_booking between the user, flight, airport, and payment

WEAK ENTITY

Train classes and Airport Classes are weak entities because their existence is dependent on train and flight entities, respectively.

All three follow the same logic:
Booking a ticket includes the user, the train/bus/flight that the user wishes to book, the source and destination stations/bus stops/airports to which the user wishes to go, and the payment for the ticket.

CHANGES MADE

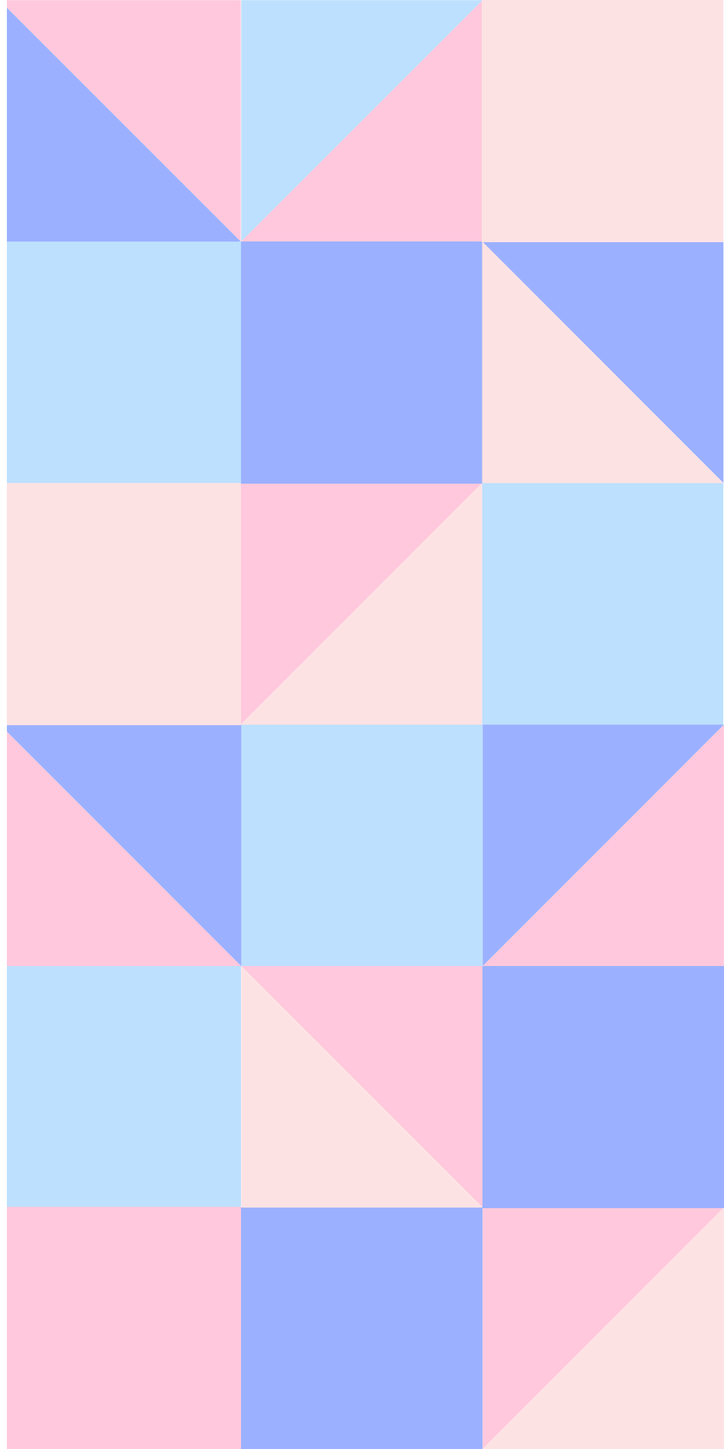
- **FLIGHT SCHEMA:**

There was a table Lands_at that was a bit redundant in our schema, so we removed it and instead added airports where the flight lands to the table Flights itself.

- **AGENT LOGIN:**

We created a login for users. The only difference between agent login and client login was the price multiplier. When a user uses an agent to book a ticket, he has to pay the price more than what he has to pay through user login. So we removed it.

- **USERS CAN NOW CANCEL TICKETS**
- **ADMIN CAN NOW CANCEL A TRAIN/BUS/FLIGHT FOR A PARTICULAR DAY**
- **CREATED AN ADMIN PAGE**



FUNCTIONS USED

FUNCTION TO FIND THE FIRST STATION FOR A PARTICULAR TRAIN

```
delimiter $$
create function trainStarting(train_id int)
 returns int
    reads SQL data
    deterministic
 begin
  declare n int;
        select ts.station_id from Train_stops ts where ts.train_id=train_id
        order by ts.arrival_day, ts.arrival_hour, ts.arrival_day limit 1 into n;
        return n;
 end $$
delimiter ;
```

FUNCTION TO FIND THE LAST STATION FOR A PARTICULAR TRAIN

```
delimiter $$
create function trainEnding(train_id int)
 returns int
    reads SQL data
    deterministic
 begin
  declare n int;
        select ts.station_id from Train_stops ts where ts.train_id=train_id
        order by ts.arrival_day DESC, ts.arrival_hour DESC, ts.arrival_minute
        DESC limit 1 into n;
        return n;
 end $$
delimiter ;
```

FUNCTION TO TELL THE NUMBER OF BOOKINGS POSSIBLE FOR A SELECTED TRAIN AND CLASS

```
delimiter $$
create function count_of1(train_id int, class_type varchar(20), train_date date)
 returns int
    reads SQL data
    deterministic
    begin
  declare n int;
        select count(*) from Train_booking where Train_booking.train_id=train_id AND
        Train_booking.train_class=class_type AND Train_booking.train_date=train_date into n;
        return n;
 end $$
delimiter ;
```

FUNCTION TO TELL THE MAXIMUM NUMBER OF BOOKINGS POSSIBLE FOR A SELECTED BUS

```
delimiter $$
create function count_of_bus(bus_id int, bus_date date)
 returns int
    reads SQL data
    deterministic
    begin
        declare n int;
        select count(*) from Bus_booking where Bus_booking.Bus_id=Bus_id
        AND Bus_booking.bus_date=bus_date into n;
        return n;
 end $$
delimiter ;
```

QUERIES

Show all kids(<18 age) traveling in the bus for a particular date.

#	first_name	last_name	age	ticket_ID	source_ID	destination_ID
1	kid	1	5	22	202	204
2	kid	2	6	23	202	204

Inflation - create a view for updated price_per_km for all trains twice
Update not view, input class

train_ID	train_type	price_per_km
1001	Garib Rath	9
1002	Sampark Kranti	3
1003	Humsafar	12
1004	Kavi Guru	6
1005	Rajdhani	15
1006	Rajdhani	15
1007	Garib Rath	9
1008	Shatabdi	18
1009	Shatabdi	18
1010	Humsafar	12
1011	Humsafar	12

Given a bus, and 2 bus stops, print details of all those users traveling between those 2 stops along with their payment details.

#	user_ID	first_name	last_name	ticket_ic	payment_ic	amount
1	3	Siya	Garg	2	69	30
2	6	Srijan	Arora	3	70	30
3	10	Srijan	Arora	4	71	30
4	9	Srijan	Arora	5	72	30
5	8	Srijan	Arora	6	73	30
6	9	Srijan	Arora	7	74	30
7	7	Srijan	Arora	8	75	30
8	5	Srijan	Arora	9	76	30
9	6	Srijan	Arora	10	77	30
10	9	Srijan	Arora	11	78	30
11	7	Srijan	Arora	12	79	30
12	6	Srijan	Arora	13	80	30

User inputs time of arrival and station, print all trains traveling away from that station and having departure time after the arrival time of user
(Eg: station-102,
arrival time-05:10)

#	train_ID	train_type
1	1011	Humsafar
2	1026	Humsafar

EMBEDDED QUERIES

Printing the booked ticket details by taking ticketID that have not yet been canceled for -

Train

TICKET DETAILS

Ticket ID	Train No.	Train Name	Train Class	Source ID	Departure time	Destination ID	Arrival time	Date	Ticket Price
12	1001	Garib Rath	AC1	101	00:00	105	02:00	2022-04-19	900

Bus

TICKET DETAILS

Ticket ID	Bus No.	Bus Type	Source ID	Departure time	Destination ID	Arrival time	Date	Ticket Price
19	2021	Sleeper	203	13:55	206	14:30	2022-04-19	30

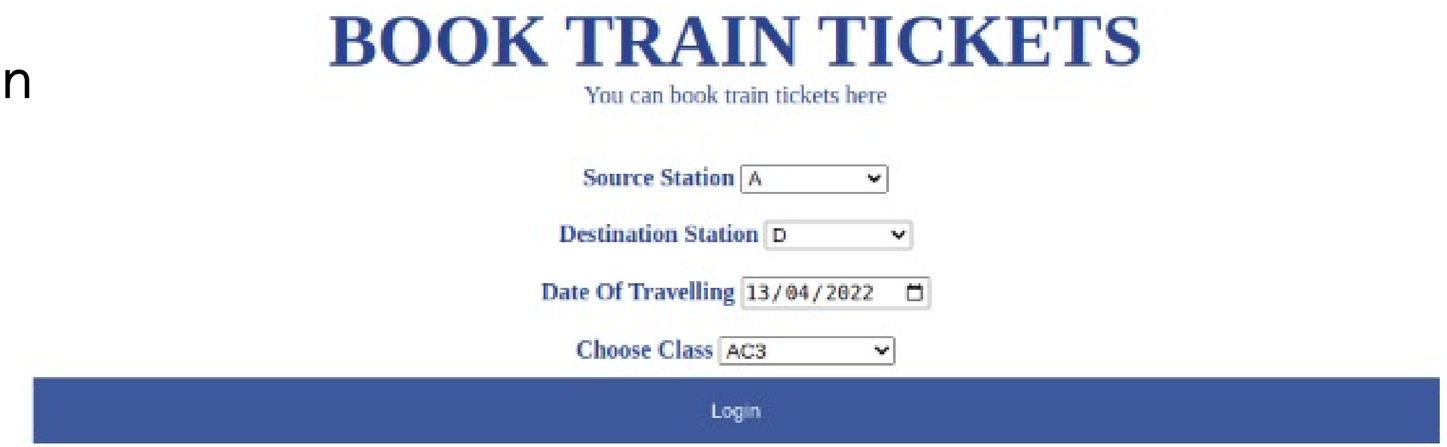
Flight

TICKET DETAILS

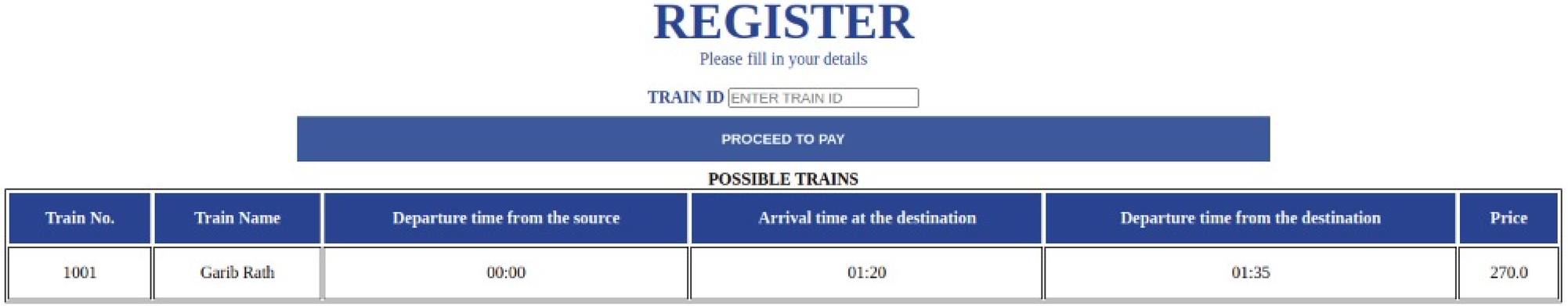
Ticket ID	Flight No.	Flight Name	Flight Class	Source ID	Departure time	Destination ID	Arrival time	Date	Ticket Price
1	3009	Etihad	First	303	07:00	305	11:30	2022-04-13	14700

Printing the available transport ID, price, timings for the following selected parameters with available seats -

a. Mode of transport
b. Source and Destination
c. Date of travel
d. Class



TRAINS:



BUS:

BOOK BUS TICKETS

You can book train tickets here

Depart From B

Going To E

Departure Date 06/04/2022

Search Bus

REGISTER

Please fill in your details

BUS ID ENTER BUS ID

PROCEED TO PAY

POSSIBLE BUSES

Bus No.	Bus Name	Departure time from the source	Arrival time at the destination	Departure time from the destination	Price
2001	AC	00:35	02:00	02:05	275

FLIGHTS:

BOOK FLIGHT TICKETS

You can book train tickets here

Depart From C

Destination Station E

Date Of Travelling 05/04/2022

Choose Class Business

Search Flight

REGISTER

Please fill in your details

TRAIN ID ENTER TRAIN ID

PROCEED TO PAY

POSSIBLE TRAINS

Flight No.	Flight Name	Departure time	Arrival time	Price
3009	Etihad	07:00	11:30	19600

TICKET DETAILS

Ticket ID	Train No.	Train Name	Train Class	Source ID	Departure time	Destination ID	Arrival time	Date	Ticket Price
53	1001	Garib Rath	AC3	101	00:00	106	02:30	2022-03-31	1800

CANCEL TICKET

You can cancel your tickets here

Vehicle TRAIN

Ticket ID 53

Submit

#	train_id	source_id	destination_id	user_id	ticket_id	payment_id	train_date	train_c
38	1010	103	106	3	39	55	2022-04-15	AC2
39	1010	103	106	7	40	56	2022-04-15	AC2
40	1010	103	106	3	41	57	2022-04-15	AC2
41	1010	103	106	6	42	58	2022-04-15	AC2
42	1010	103	106	6	43	59	2022-04-15	AC2
43	1010	103	106	8	44	60	2022-04-15	AC2
44	1010	103	106	7	45	61	2022-04-15	AC2
45	1010	103	106	9	46	62	2022-04-15	AC2
46	1010	103	106	10	47	63	2022-04-15	AC2
47	1010	103	106	8	48	64	2022-04-15	AC2
48	1010	103	106	4	49	65	2022-04-15	AC2
49	1010	103	106	5	50	66	2022-04-15	AC2
50	1001	101	106	3	53	91	2022-03-31	AC3

#	train_id	source_id	destination_id	user_id	ticket_id	payment_id	train_date	train_class
37	1010	103	106	2	38	54	2022-04-15	AC2
38	1010	103	106	3	39	55	2022-04-15	AC2
39	1010	103	106	7	40	56	2022-04-15	AC2
40	1010	103	106	3	41	57	2022-04-15	AC2
41	1010	103	106	6	42	58	2022-04-15	AC2
42	1010	103	106	6	43	59	2022-04-15	AC2
43	1010	103	106	8	44	60	2022-04-15	AC2
44	1010	103	106	7	45	61	2022-04-15	AC2
45	1010	103	106	9	46	62	2022-04-15	AC2
46	1010	103	106	10	47	63	2022-04-15	AC2
47	1010	103	106	8	48	64	2022-04-15	AC2
48	1010	103	106	4	49	65	2022-04-15	AC2
49	1010	103	106	5	50	66	2022-04-15	AC2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

EMBEDDED QUERIES IN ADMIN

GIVE ALL THE BOOKINGS WITH GIVEN SOURCE, DESTINATION AND TRAVEL DATE

TOTAL PAYMENTS

TICKET ID	VEHICLE ID	USER ID	CLASS	PAYMENT ID	AMOUNT
23	1010	2	AC2	39	120
24	1010	3	AC2	40	120
25	1010	4	AC2	41	120
26	1010	4	AC2	42	120
27	1010	3	AC2	43	120
28	1010	7	AC2	44	120
29	1010	7	AC2	45	120
30	1010	4	AC2	46	120
31	1010	8	AC2	47	120
32	1010	8	AC2	48	120
33	1010	6	AC2	49	120
34	1010	4	AC2	50	120
35	1010	5	AC2	51	120

GIVE THE ROUTE OF A TRAIN IN INCREASING ORDER OF ARRIVAL TIME

VEHICLE PATH

Vehicle ID	Station/Stop/Airport ID	Arrival time	Departure time
2021	209	11:00	11:10
2021	208	11:55	12:05
2021	207	12:45	12:55
2021	203	13:50	13:55
2021	206	14:30	14:45
2021	205	15:45	16:00

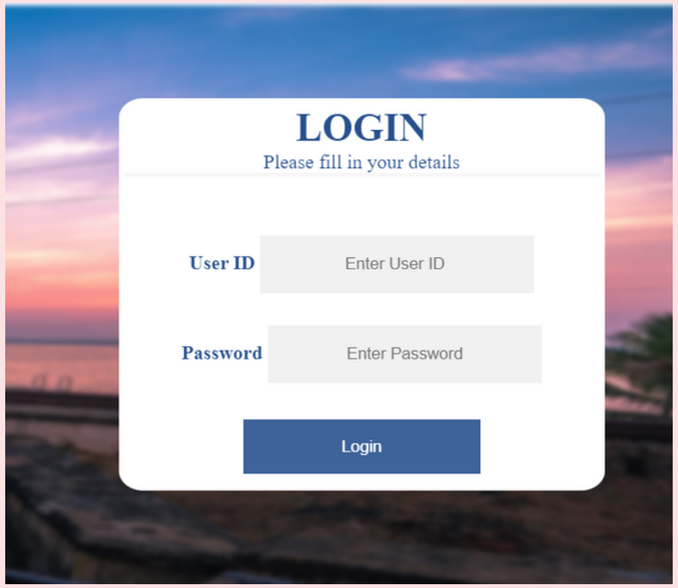
GIVEN A TRAIN AND A DATE, GIVE ALL THE BOOKINGS THAT HAVE BEEN DONE

TOTAL PAYMENTS

VEHICLE ID	TRAVEL DATE	TOTAL SUM	PAYMENT TYPE
2021	2022-04-19	120	CASH
2021	2022-04-19	60	UPI
2021	2022-04-19	240	NET BANKING
2021	2022-04-19	180	BANK TRANSFER

FUNCTIONALITY SUPPORTED BY UI

CLIENT:



TICKET DETAILS

Please fill in your details

Ticket ID Enter Ticket ID

Sign In

BOOK TRAIN TICKETS

You can book train tickets here

Source Station Depart From

Destination Station Depart From

Date Of Travelling dd-mm-yyyy

Choose Class Select the class

Login

CANCEL TICKETS

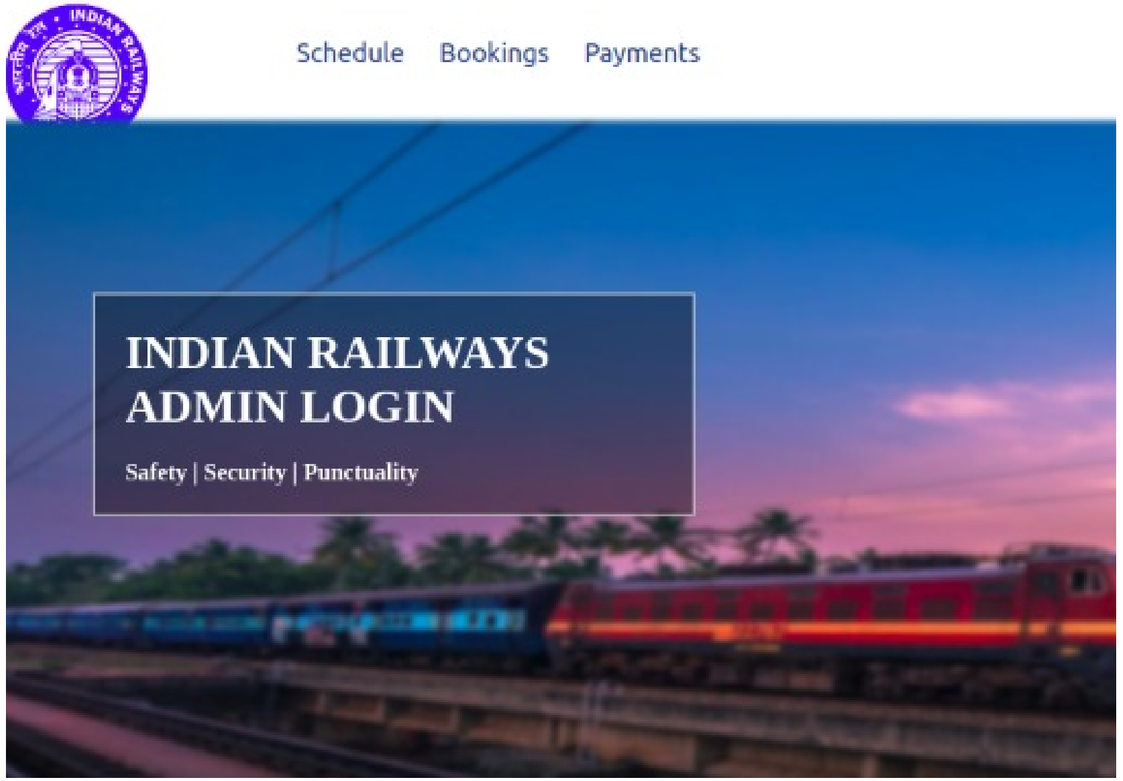
You can cancel tickets here

MODE OF TRANSPORT TRAIN

Ticket ID Enter Ticket ID

Login

ADMIN



VEHICLE SCHEDULE

STOP SCHEDULE

QUERY OPTIMIZATION

SQL is an advanced database language that optimizes the queries on its own while executing. To facilitate the query optimization process, we kept the following things in mind while writing our queries:

1. Performing **selection** as early as possible: reduces the size of relations being joined
2. **Execution plan**: A systematic step-by-step execution of primitive operations for fetching data from the database
3. To **join** only those tables which were **necessary** for all data needed. (minimize the number of joins)
4. Multi query optimization: find the best overall plan for a set of queries, and make use of sharing of common subexpressions.

For **Parametric query optimization**, optimization is done at **runtime** by the optimizer in the following way:

- The optimizer generates a **set of plans**, optimal for different values
- If the optimizer decides that the same plan is likely to be optimal for all parameter values, it **caches the plan and reuses it**, else reoptimizing each time
- **avoid restarting** to not lose previously cached data.

We tried our best to optimize the queries

Here is an **example** of an optimized query that we have used in the functioning of our website:

```
Select Train.train_ID, Train.train_type from Train inner join
Train_stops on Train.train_ID = Train_stops.train_ID where
Train_stops.station_ID=102 and trainEnding(Train.train_ID)!=102
and ((Train_stops.arrival_hour>5) OR (Train_stops.arrival_hour=5 AND
Train_stops.arrival_minute>=10));
```

GRANTS

The **admin** of our IRCTC website and database has been **granted** the access to -

ADMIN:

select and delete Users
select delete and insert Payments
select and update info on Trains, Buses, Flights
select TrainStops, Stations, TrainDistanceFrom
select and update TrainClasses, FlightClasses
select and delete TrainBookings(Tickets)
select TrainStops, Stations, TrainDistanceFrom
select BusStopsAt, BusStops, BusDistanceFrom
select and delete BusBookings(Tickets)
select Airport
select and delete FlightBookings(Tickets)

CLIENT:

The **client** on our IRCTC website has been **granted** the access to -

select and insert in User

select, insert and delete Payment

select a train, Station, TrainStops, TrainDistanceFrom

select, insert and delete TrainBooking

select a Bus, BusStop, BusStopsAt, BusDistanceFrom

select, insert and delete BusBooking

select a Airport, Flight, FlightClasses

select, insert and delete FlightBooking

VIEWS

VIEWS FOR SUM OF ALL EARNINGS FOR:

TRAIN:

```
create view TrainAllPayments as
select p.payment_type, sum(p.amount)
from Train_booking tb join Payment p on tb.payment_id=p.payment_id
group by p.payment_type;
```

BUS:

```
create view BusAllPayments as
select p.payment_type, sum(p.amount)
from Bus_booking bb join Payment p on bb.payment_id=p.payment_id
group by p.payment_type;
```

FLIGHT:

```
create view FlightAllPayments as
select p.payment_type, sum(p.amount)
from Flight_booking fb join Payment p on fb.payment_id=p.payment_id
group by p.payment_type;
```

VIEW FOR PRINTING ALL THE PEOPLE BELONGING TO SPECIAL CATEGORIES IE WOMEN, SENIOR CITIZENS AND INFANTS

```
create view specialPeople as
select * from User
where User.age<2 OR User.age>=60 OR User.gender='F';
```

VIEWS FOR SUM OF ALL LOSS FROM CANCELLED TICKETS FOR:

TOTAL PAYMENTS

TRANSACTION TYPE	TOTAL AMOUNT
CASH	620
UPI	60
NET BANKING	240
BANK TRANSFER	305

TRAIN:

```
create view TrainAllCancelledPayments as
 select p.payment_type, sum(p.amount)
 from Train_cancelled tc join Payment p on tc.payment_id=p.payment_id
 group by p.payment_type;
```

BUS:

```
create view BusAllCancelledPayments as
 select p.payment_type, sum(p.amount)
 from Bus_cancelled bc join Payment p on bc.payment_id=p.payment_id
 group by p.payment_type;
```

FLIGHT:

```
create view FlightAllCancelledPayments as
 select p.payment_type, sum(p.amount)
 from Flight_cancelled fc join Payment p on fc.payment_id=p.payment_id
 group by p.payment_type;
```

TRIGGERS

TO DELETE CANCELLED TICKETS FROM MAIN RECORDS AND THEM TO CANCELLED TICKET TABLE FOR:

TRAIN:

```
delimiter $$
create trigger TrainTicketCancel after delete on Train_booking
for each row
begin
    insert into train_cancelled values (old.train_id,
    old.source_id, old.destination_id, old.user_id, old.ticket_id,
    old.payment_id, old.train_date, old.train_class);
end$$
delimiter ;
```

BUS:

```
delimiter $$
create trigger BusTicketCancel after delete on Bus_booking
for each row
begin
    insert into Bus_cancelled values
    (old.bus_id, old.source_id, old.destination_id,
     old.user_id, old.ticket_id, old.payment_id, old.bus_date);
end$$
delimiter ;
```

FLIGHT:

```
delimiter $$
create trigger FlightTicketCancel after delete on Flight_booking
for each row
begin
    insert into Flight_cancelled values
    (old.flight_id, old.source_id, old.destination_id,
     old.user_id, old.ticket_id,old.payment_id, old.flight_date,
     old.flight_class);
end$$
delimiter ;
```

TO CALCULATE THE AGE OF A USER BASED ON THE INPUT (DATE OF BIRTH) -

```
delimiter $$
create trigger calcAge before insert on user
for each row
begin
 set new.age = 2022 - year(new.DOB);
end$$
delimiter ;
```

INDEXING

select * from Train_stops where train_id='{0}' order by arrival_day, arrival_hour, arrival_minute
The index used here is a composite index: arrival_day, arrival_hour and arrival_minute

select * from Bus_stops_at where bus_id='{0}' order by arrival_day, arrival_hour, arrival_minute
The index used here is a composite index: arrival_day, arrival_hour and arrival_minute

The other queries use the primary keys of the tables that they are being run on. The primary keys are used to create the primary index, which is what serves as our index. Primary keys have been defined in the DDL submitted

THANK YOU

MORE QUERIES AND STUFF IN DBMS.PDF