TAP CAB

Team 114

Prepared by Niharika Singh(2021545) and Divya Raj Singh(2021528)

PROJECT SCOPE

My means of our application we have tried to implement numerous and some of the most essential functionality and features for booking a cab online. We have provided easy and user friendly interface of our application. Some essential features include vehicle management, safety SOS feature, managing payments online via various means to help user friendliness, adding stops in between destination and car music integration and maintaining the privacy of the user. We try to ensure that our system can handle many requests at the same time so that the user can get a cab booked with just a tap by our application TAP CAB!

TECH STACK

FRONT END

1. HTML
2. CSS
3. Python
4. Flask

BACK END

1. mySQL for Database Management

FUNCTIONAL REQUIREMENTS

1. Driver and Vehicle Management
2. Integration with Google Maps
3. Security for sensitive user data
4. Payment and Booking Management
5. SOS and Emergency Features
6. Driver confirms trip

7. The customer books the cab

ENTITIES

1. Admin (Admin_Id, first name, mid name, Last name, phone number, email, address)
2. Person (ID, Name, first name, last name, phone no., email, address, house no, street, city, state, Pincode, general information)
3. Driver (ID, preference type, rating)
4. Cab (Cab id, Cab_type, Registration_No, fuel)
5. Customer (ID, password)
6. Trip (trip id, fare details, estimated time, pick up, destination)
7. Payment(transaction id, type, date) {weak}
8. Map(Cab id, location) {weak}
9. SOS(Cab id, calling option, help guidelines) {weak}
10. Review(CID, DID, stars, comment, suggestion) {weak}
11. Customer care(ID, skills)

RELATIONSHIPS

1. Books (Date, time)-> customer books cab
2. Drives -> Driver drives cab
3. Confirm (Date, time) -> driver confirm trip
4. Supervise -> driver supervise driver
5. Select -> customer select cab
6. Integrated -> cab system integrated with cab
7. Get location -> trip get location from map
8. Record -> trip record payment
9. Verifies -> Admin verifies trip
10. Confirms -> Admin confirms trip
11. Supervises -> Admin supervises drivers

CONSTRAINTS

1. A Driver can’t belong to more than one cab
2. A Driver can only confirm a trip

SCOPE

The company Tap Cab has decided to open a facility to provide cabs for booking. It plans to hire the drivers, and while hiring, it takes the driver's personal information, such as the driver’s name, phone number, and email address. It gives a unique driver id to each individual. The company also provides cabs to each individual with a unique id for each cab based on the information they provided and which type of cabs they can drive with the cab’s registration number and type of fuel. Different cabs have been modified with integrated maps, SOS systems for customers' safety, and music systems for customer enjoyment. The company has provided the facility of vehicle management for each cab. The company provides a user-friendly interface for users to book cabs according to their needs. Each customer must sign up and provide their name, phone number, address, and password. Each customer gets a unique customer id. A customer first has to choose the type of cab and the destination. After this, they must select the payment options, online or offline. A customer needs to wait until the driver confirms the trip, including the pickup location, destination location, and fair details; this trip goes to all nearby drivers. The driver gets information regarding the trip, such as pick-up location, destination location, estimated time, and fair details. A driver can accept or

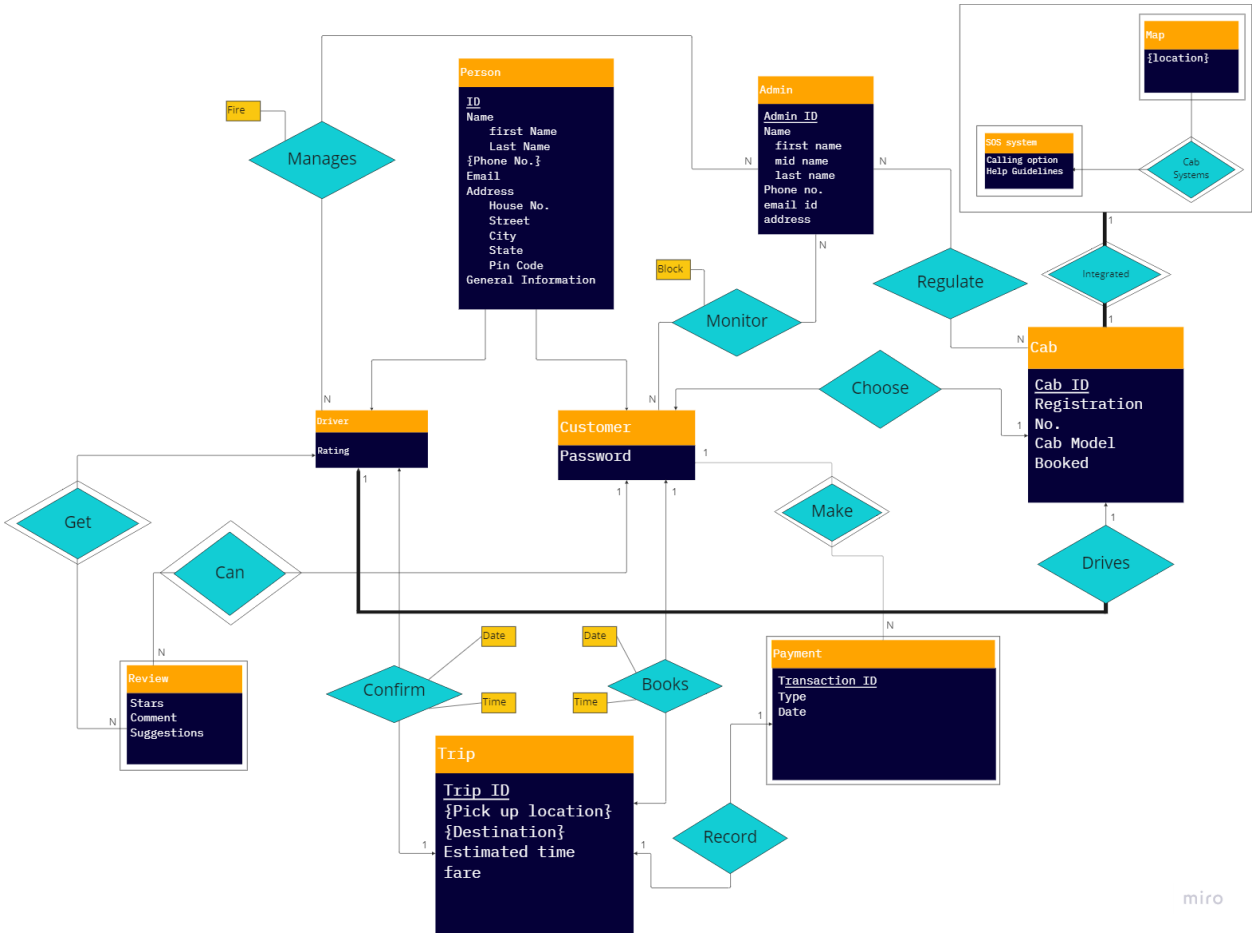
reject the trip according to his convenience. Each trip will get a unique trip id. After the trip, each customer can rate the driver and cab.

Project 2 : ER diagram

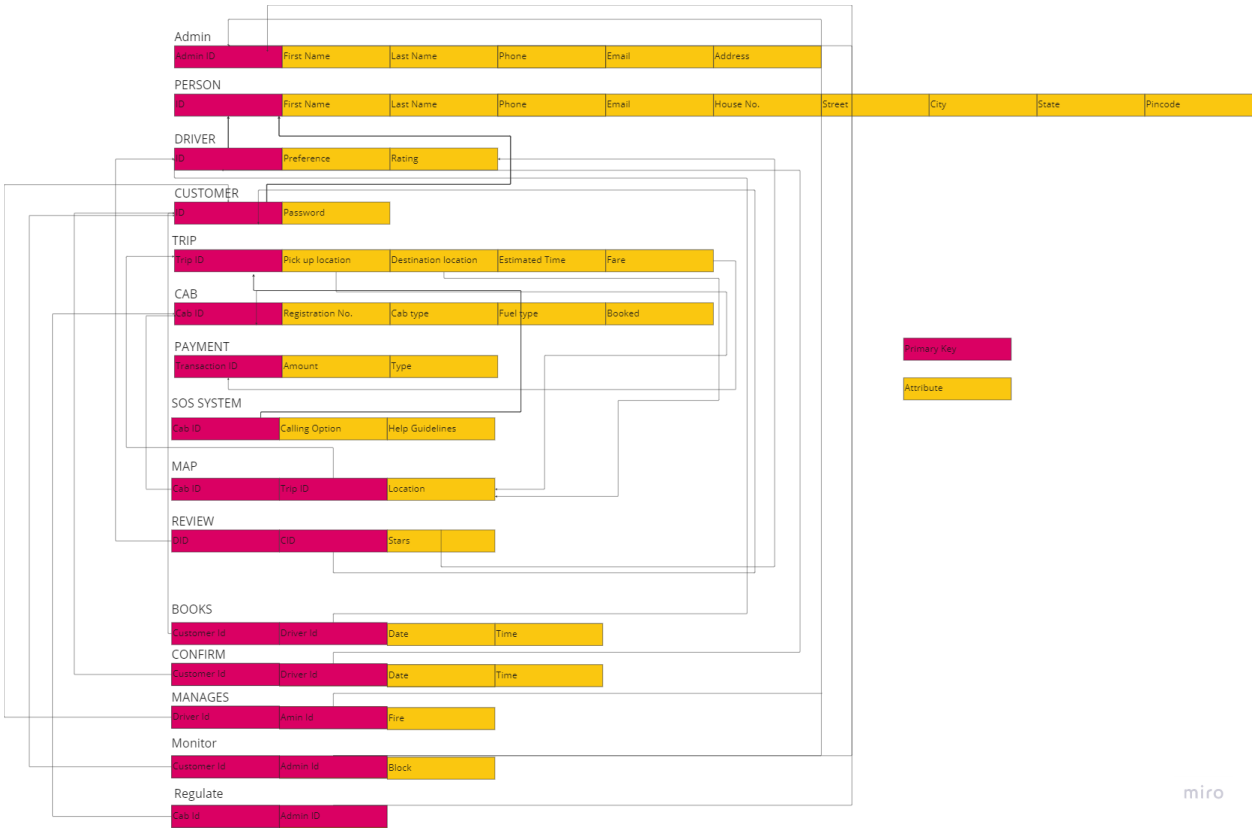
Documentation:

1. Admin can access the whole data
2. Driver can access only the own data, allotted cab data and some customer data
3. One driver can only drive one cab
4. One driver can only confirm one trip at a time with specific date and time
5. A driver must drive a car
6. Admin verifies the trip
7. Customer can access only the own data, some of the driver and cab data
8. A customer can choose multiple cab
9. One cab is associated with one customer
10. A customer can select any payment options
11. One transaction is associated with one customer
12. One customer can book only one trip at a time with specific date and time
13. A customer can make multiple payments
14. One payment is linked to only one customer
15. A trip can only have one customer
16. A trip can record one payment
17. Admin supervises driver
18. A customer can give multiple reviews
19. A driver can get a review
20. A review is linked to only one driver
21. A driver can supervise other drivers
22. A cab must have sos and map system integrated
23. Trip can access locations through map

24. Admin confirms customer



Relational Schema



Documentation Project 3:

Integrity Constraints:

1. Person:
    a. Id (varchar) (primary key) (Not null)
    b. First name (varchar)
    c. Last name (varchar) (not null)
    d. Phone (long int)
    e. Email (varchar) (not null)
    f. House no (varchar)

    g. Street (varchar)
    h. City (varchar) (not null)
    i. State (varchar) (not null)
    j. Pincode (int) (not null)

2. Driver
    a. Id (varchar) (foreign key) (not null)
    b. Preference (varchar)
    c. Rating (int)
3. Customer
    a. Id (varchar) (foreign key)
    b. Password (varchar)
4. Trip
    a. Trip id (varchar)

Assumptions :

Admin

1. Multiple admins manage multiple drivers
2. Multiple admins monitor multiple cabs
3. Multiple admins regulate multiple customers

Person

1. One person can either be a driver or a customer

Driver

1. One driver can drive one cab
2. Every driver must drive one cab
3. One driver can confirm one trip
4. A driver can get multiple reviews

Customer

1. One customer can book one trip
2. One customer can choose one car

3. One customer can make multiple payments
4. One customer can give multiple reviews

Trip

1. One trip can be booked by one customer
2. One trip can be confirmed by one driver
3. One trip can record one payment

Project deadline 3: Mistakes

1. phone no. unique
2. fare int not null
3. price not negative

Project 4:

SQL queries

SELECT C.CustomerId, D.DriverId, T.TripId, T.DriverId, T.CustomerId FROM customer AS C, driver AS d, trip AS T

WHERE C.CustomerId = T.CustomerId AND D.DriverId = T.DriverId;

SELECT * FROM Driver WHERE Status = 'Unoccupied';

SELECT * FROM books;

UPDATE Driver

set Status = 'Occupied'

WHERE DriverId IN (SELECT Driverid FROM Confirm);

SELECT * FROM Driver;

```
SELECT * FROM Confirm c;

UPDATE Trip
SET Status = 'Successful'
WHERE TripId IN (SELECT TripId FROM Confirm);

SELECT * FROM Trip t;



UPDATE Driver
SET Status = 'Unoccupied'
WHERE DriverId IN (SELECT t.DriverId FROM Trip t WHERE t.Status = 'Successful');

SELECT t.DropLoc,d.Location FROM Trip t,Driver AS d WHERE t.DriverId = d.DriverId;

UPDATE Driver d
JOIN Trip t ON d.DriverId = t.DriverId
SET d.Location = t.DropLoc
WHERE t.Status = 'Successful';

SELECT * FROM Driver;

SELECT DriverId, COUNT(*) AS trip_Count FROM trip
```

GROUP BY DriverId;

SELECT CustomerId, SUM(Fare) AS total FROM payment

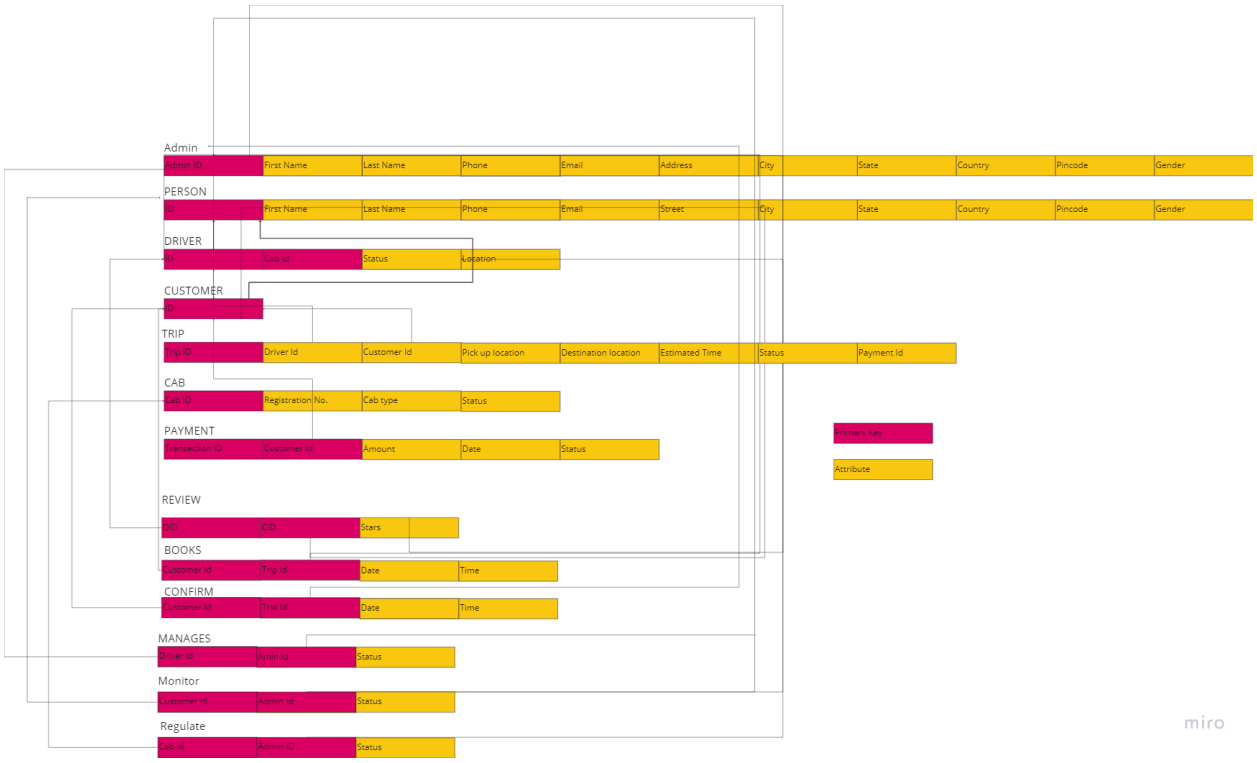
GROUP BY CustomerId

ORDER BY total DESC

LIMIT 10;

SELECT * FROM Driver

INNER JOIN Cab c ON Driver.CabId = c.CabId;



OLAP queries

1

```
SELECT
    YEAR(Booking.booking_time) AS Year,
    Cab.type AS CabType,
    SUM(Payment.amount) AS TotalRevenue
FROM
    Booking
    JOIN Cab ON Booking.cab_id = Cab.cab_id
    JOIN Payment ON Booking.booking_id = Payment.booking_id
WHERE
    YEAR(Booking.booking_time) = 2023
GROUP BY
    YEAR(Booking.booking_time),
    Cab.type WITH ROLLUP
```

2

```
SELECT
    Cab.type AS CabType,
    YEAR(Booking.booking_time) AS Year,
    QUARTER(Booking.booking_time) AS Quarter,
    SUM(Payment.amount) AS TotalRevenue,
```

```
    AVG(SUM(Payment.amount)) OVER (PARTITION BY Cab.type,
YEAR(Booking.booking_time) ORDER BY QUARTER(Booking.booking_time) ROWS
BETWEEN 2 PRECEDING AND 3 FOLLOWING) AS SixMonthMovingAverage,

    RANK() OVER (PARTITION BY Cab.type, YEAR(Booking.booking_time) ORDER BY
SUM(Payment.amount) DESC) AS Ranking

FROM

    Booking

    JOIN Cab ON Booking.cab_id = Cab.cab_id

    JOIN Payment ON Booking.booking_id = Payment.booking_id

GROUP BY

    Cab.type,

    YEAR(Booking.booking_time),

    QUARTER(Booking.booking_time);
```

3

```
SELECT

    Cab.type AS CabType,

    MONTH(Payment.payment_time) AS Month,

    COUNT(DISTINCT Booking.customer_id) AS UniqueCustomers,

    SUM(Payment.amount) AS TotalRevenue

FROM

    Booking

    JOIN Cab ON Booking.cab_id = Cab.cab_id

    JOIN Payment ON Booking.booking_id = Payment.booking_id
```

```
WHERE
    YEAR(Payment.payment_time) = 2023
GROUP BY
    Cab.type,
    MONTH(Payment.payment_time)
```

4

```
SELECT
    Driver.driver_id AS DriverId,
    Driver.name AS DriverName,
    YEAR(Booking.booking_time) AS Year,
    SUM(Booking.fare) AS Fare
FROM
    Booking
    JOIN Cab ON Booking.cab_id = Cab.cab_id
    JOIN Driver ON Cab.driver_id = Driver.driver_id
GROUP BY
    Driver.driver_id,
    Driver.name,
    YEAR(Booking.booking_time)
    WITH ROLLUP
HAVING
    DriverId IS NOT NULL
    AND Year IS NOT NULL;
```

5

```
SELECT
    Customer.customer_id AS CustomerId,
    Customer.name AS CustomerName,
    Driver.driver_id AS DriverId,
    Driver.name AS DriverName,
    YEAR(Booking.booking_time) AS Year,
    COUNT(*) AS Bookings
FROM
    Booking
    JOIN Cab ON Booking.cab_id = Cab.cab_id
    JOIN Driver ON Cab.driver_id = Driver.driver_id
    JOIN Customer ON Booking.customer_id = Customer.customer_id
GROUP BY
    Customer.customer_id,
    Customer.name,
    Driver.driver_id,
    Driver.name,
    YEAR(Booking.booking_time)
    WITH ROLLUP
HAVING
    CustomerId IS NOT NULL
    AND DriverId IS NOT NULL
```

```
    AND Year IS NOT NULL;
```

Triggers

1

```
DELIMITER $$

CREATE TRIGGER fill_trip_table

AFTER update ON Booking

FOR EACH ROW

BEGIN

    IF NEW.status = 'CONFIRMED' THEN

        SET @distance_value := (SELECT distance FROM Location WHERE start_location =
NEW.pickup_location AND end_location = NEW.destination);

        SET @fare_value := (SELECT fare FROM Location WHERE start_location =
NEW.pickup_location AND end_location = NEW.destination);

        INSERT INTO Trip (booking_id, driver_id, start_time, distance, fare)

        VALUES (NEW.booking_id, (SELECT driver_id FROM Confirm WHERE booking_id =
NEW.booking_id), NOW(), @distance_value, @fare_value);

    END IF;

END$$

DELIMITER ;
```

2

```
DELIMITER $$
```

```
CREATE TRIGGER update_driver_rating
AFTER INSERT ON Review
FOR EACH ROW
BEGIN
    UPDATE Driver
    SET rating = (SELECT AVG(rating) FROM Review WHERE driver_id = NEW.driver_id)
    WHERE driver_id = NEW.driver_id;
END$$
DELIMITER ;
```

TRANSACTIONS

```
-- Transactions 1
START TRANSACTION;

-- Insert new booking
INSERT INTO Booking (customer_id, pickup_location, destination, fare, booking_time)
VALUES (5, '11th Floor', 'PO Box 34715', 359, NOW());

-- Find available cab
SET @free_cab_id := (
        SELECT cab_id
        FROM Cab
        WHERE cab_status = 'free'
        ORDER BY RAND()
        LIMIT 1
);

-- Update cab status to 'assigned'
```

```
UPDATE Cab
SET cab_status = 'BOOKED'
WHERE cab_id = @free_cab_id;

-- Find available driver for cab
SET @free_driver_id := (
        SELECT driver_id
        FROM Driver
        WHERE driver_id NOT IN (SELECT driver_id FROM Confirm WHERE confirm_time
IS NULL)
        AND driver_id IN (SELECT driver_id FROM Cab WHERE cab_status = 'BOOKED')
        ORDER BY RAND()
        LIMIT 1
);

-- Insert confirm record for booking and driver
INSERT INTO Confirm (booking_id, driver_id, confirm_time)
VALUES (LAST_INSERT_ID(), @free_driver_id, NOW());

-- Update cab record with driver id
UPDATE Cab
SET driver_id = @free_driver_id
WHERE cab_id = @free_cab_id;
```

```
rollback;



-- Transaction 2

START TRANSACTION;


-- Get the list of all available cabs and their distance from the pickup location

SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating,
Location.distance

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

INNER JOIN Location ON Location.start_location = 'Room 1908' AND
Location.end_location = 'Room 485'

WHERE Cab.cab_status = 'free';


-- Assume that the customer has chosen cab_id = 1, and get the details of the cab and
driver

SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

WHERE Cab.cab_id = 1;


-- Get the fare for the trip based on the distance between pickup location and destination

SELECT fare
```

```
FROM Location

WHERE start_location = 'Room 1908' AND end_location = 'Room 485';


-- Create a new booking record for the customer

INSERT INTO Booking (customer_id, cab_id, pickup_location, destination, fare,
booking_time)

VALUES (1, 1, 'Room 1908', 'Room 485', 899, NOW());


-- Update the cab status to 'booked'

UPDATE Cab SET cab_status = 'booked' WHERE cab_id = 1;


-- Insert the confirmed booking record for the driver

INSERT INTO Confirm (booking_id, driver_id, confirm_time)

VALUES (LAST_INSERT_ID(), 2, NOW());


-- Update the driver's location to the pickup location

UPDATE dri_current SET dri_loc = 'Room 1908' WHERE driver_id = 2;


-- Insert a new trip record

INSERT INTO Trip (booking_id, driver_id, start_time, distance, fare)

VALUES (LAST_INSERT_ID(), 2, NOW(), 26, 899);


-- Prompt the customer to make a payment for the trip

SELECT Booking.booking_id, Booking.fare
```

```
FROM Booking

WHERE Booking.customer_id = 1 AND Booking.status = 'BOOKED';


-- Assume that the customer has made a payment for the trip, update the payment record

UPDATE Payment SET amount = 899, payment_time = NOW(), payment_status =
'PAYMENT DONE'

WHERE Payment.booking_id = 1;


-- Update the booking status to 'COMPLETED'

UPDATE Booking SET status = 'COMPLETED' WHERE booking_id = 1;


-- Update the cab status to 'free'

UPDATE Cab SET cab_status = 'free' WHERE cab_id = 1;


-- Insert the customer's review for the driver

INSERT INTO Review (booking_id, driver_id, customer_id, rating, comment)

VALUES (1, 2, 1, 9, 'The driver was polite and drove safely.');


rollback;
```

Conflict Serializable Schedule

T1: START TRANSACTION;

T2: START TRANSACTION;

T2: SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating, Location.distance

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

INNER JOIN Location ON Location.start_location = 'Room 1908' AND Location.end_location = 'Room 485'

WHERE Cab.cab_status = 'free';

T2: SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

WHERE Cab.cab_id = 1;

T2: SELECT fare

FROM Location

WHERE start_location = 'Room 1908' AND end_location = 'Room 485';

T2: INSERT INTO Booking (customer_id, cab_id, pickup_location, destination, fare, booking_time)

VALUES (1, 1, 'Room 1908', 'Room 485', 899, NOW());

T2: UPDATE Cab SET cab_status = 'booked' WHERE cab_id = 1;

T2: INSERT INTO Confirm (booking_id, driver_id, confirm_time)

VALUES (LAST_INSERT_ID(), 2, NOW());

T2: UPDATE dri_current SET dri_loc = 'Room 1908' WHERE driver_id = 2;

T2: INSERT INTO Trip (booking_id, driver_id, start_time, distance, fare)

VALUES (LAST_INSERT_ID(), 2, NOW(), 26, 899);

T2: SELECT Booking.booking_id, Booking.fare

FROM Booking

WHERE Booking.customer_id = 1 AND Booking.status = 'BOOKED';

T2: UPDATE Payment SET amount = 899, payment_time = NOW(), payment_status = 'PAYMENT DONE'

WHERE Payment.booking_id = 1;

T2: UPDATE Booking SET status = 'COMPLETED' WHERE booking_id = 1;

T1: INSERT INTO Booking (customer_id, pickup_location, destination, fare, booking_time)

VALUES (5, '11th Floor', 'PO Box 34715', 359, NOW());

T1: SET @free_cab_id := (

SELECT cab_id

FROM Cab

WHERE cab_status = 'free'

ORDER BY RAND()

LIMIT 1

);

T1: UPDATE Cab

SET cab_status = 'BOOKED'

WHERE cab_id = @free_cab_id;

T1: SET @free_driver_id := (

SELECT driver_id

FROM Driver

WHERE driver_id NOT IN (SELECT driver_id FROM Confirm WHERE confirm_time IS NULL)

AND driver_id IN (SELECT driver_id FROM Cab WHERE cab_status = 'BOOKED')

ORDER BY RAND()

LIMIT 1

);

T1: INSERT INTO Confirm (booking_id, driver_id, confirm_time)

T1: UPDATE Cab

SET driver_id = @free_driver_id

WHERE cab_id = @free_cab_id;

T1: Commit;

T2: INSERT INTO Review (booking_id, driver_id, customer_id, rating, comment)

VALUES (1, 2, 1, 9, 'The driver was polite and drove safely.');

T2: Commit;

T1	T2
	R(Cab)
	R(Cab)
	R(Location)
	W(Booking)
	W(Cab)
	W(Confirm)
	W(dri_current)
	W(Trip)
	R(Booking)
	W(Payment)
	W(Booking)
W(Booking)	
	W(Cab)
R(Cab)	
W(Cab)	

R(Driver)	
R(Cab)	
W(Confirm)	
W(Cab)	
Commit	
	W(Review)-
	Commit

T2- - - - - - - - - - - - - - -> T1

There is no cycle hence it is conflict Serializable

Not Conflict Serializable Schedule

T1: START TRANSACTION;

T2: START TRANSACTION;

T2: SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating, Location.distance

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

INNER JOIN Location ON Location.start_location = 'Room 1908' AND Location.end_location = 'Room 485'

WHERE Cab.cab_status = 'free';

T1: INSERT INTO Booking (customer_id, pickup_location, destination, fare, booking_time)

VALUES (5, '11th Floor', 'PO Box 34715', 359, NOW());

T2: SELECT Cab.cab_id, Cab.registration_number, Cab.type, Driver.name, Driver.rating

FROM Cab

INNER JOIN Driver ON Cab.driver_id = Driver.driver_id

WHERE Cab.cab_id = 1;

T2: SELECT fare

FROM Location

WHERE start_location = 'Room 1908' AND end_location = 'Room 485';

T1: SET @free_cab_id := (

SELECT cab_id

FROM Cab

WHERE cab_status = 'free'

ORDER BY RAND()

LIMIT 1

);

T1: UPDATE Cab

SET cab_status = 'BOOKED'

WHERE cab_id = @free_cab_id;

T2: INSERT INTO Booking (customer_id, cab_id, pickup_location, destination, fare, booking_time)

VALUES (1, 1, 'Room 1908', 'Room 485', 899, NOW());

T1: SET @free_driver_id := (

SELECT driver_id

FROM Driver

WHERE driver_id NOT IN (SELECT driver_id FROM Confirm WHERE confirm_time IS NULL)

AND driver_id IN (SELECT driver_id FROM Cab WHERE cab_status = 'BOOKED')

ORDER BY RAND()

LIMIT 1

);

T1: INSERT INTO Confirm (booking_id, driver_id, confirm_time)

VALUES (LAST_INSERT_ID(), @free_driver_id, NOW());

T2: UPDATE Cab SET cab_status = 'booked' WHERE cab_id = 1;

T1: UPDATE Cab

SET driver_id = @free_driver_id

WHERE cab_id = @free_cab_id;

T2: INSERT INTO Confirm (booking_id, driver_id, confirm_time)

VALUES (LAST_INSERT_ID(), 2, NOW());

T2: UPDATE dri_current SET dri_loc = 'Room 1908' WHERE driver_id = 2;

T2: INSERT INTO Trip (booking_id, driver_id, start_time, distance, fare)

VALUES (LAST_INSERT_ID(), 2, NOW(), 26, 899);

T1: Commit;

T2: SELECT Booking.booking_id, Booking.fare

FROM Booking

WHERE Booking.customer_id = 1 AND Booking.status = 'BOOKED';

T2: UPDATE Payment SET amount = 899, payment_time = NOW(), payment_status = 'PAYMENT DONE'

WHERE Payment.booking_id = 1;

T2: UPDATE Booking SET status = 'COMPLETED' WHERE booking_id = 1;

T2: UPDATE Cab SET cab_status = 'free' WHERE cab_id = 1;

T2: INSERT INTO Review (booking_id, driver_id, customer_id, rating, comment)

VALUES (1, 2, 1, 9, 'The driver was polite and drove safely.');

T2: Commit;

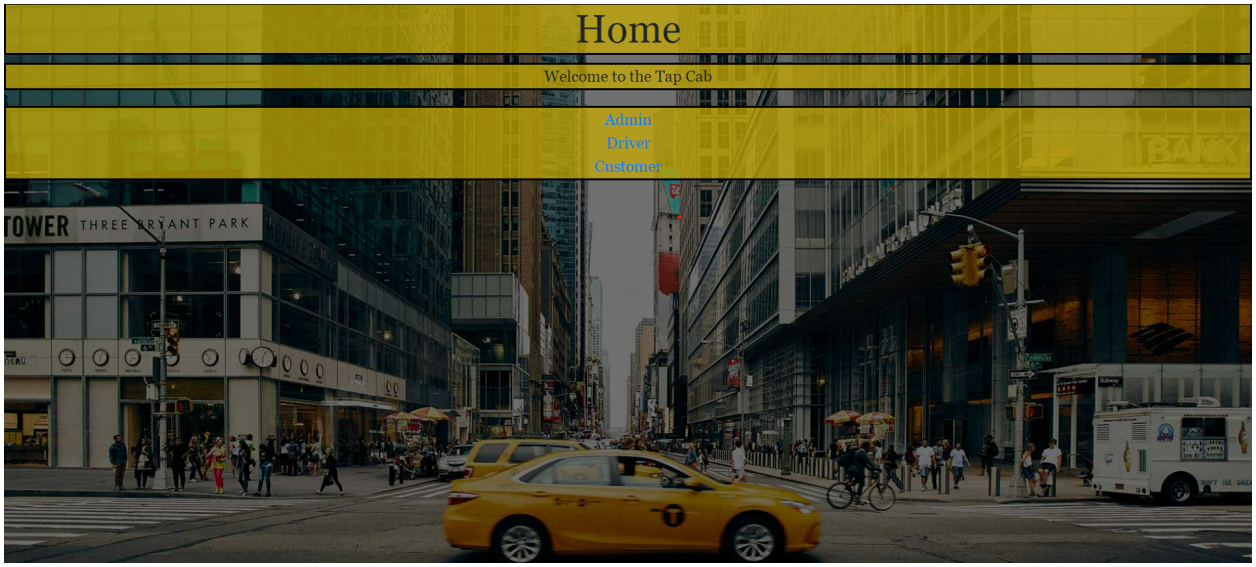
T1	T2
	R(Cab)- - - -
W(Booking)- - - - - -	
	R(Cab)
	R(Location)
R(Cab)	
W(Cab) - - - - - -	
	W(Booking) - - - – -
R(Driver)	
R(Cab)	
W(Confirm)	
	W(Cab)
W(Cab)	
	W(Confirm)
	W(dri_current)
	W(Trip)
Commit	
	R(Booking)
	W(Payment)
	W(Booking)
	W(Cab)
	W(Review)
	Commit

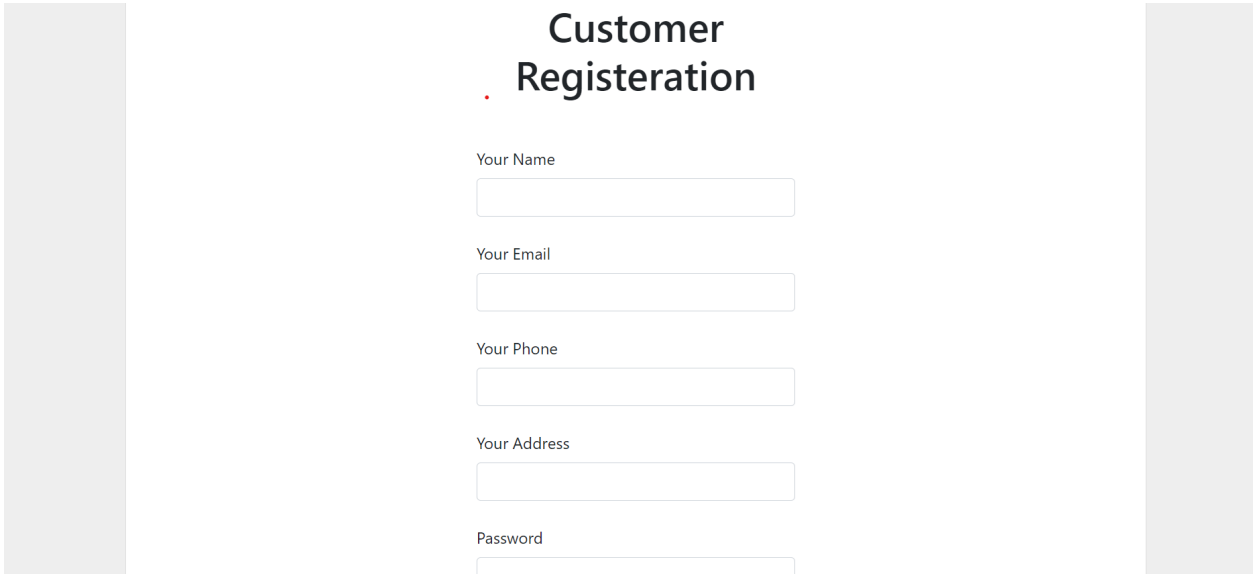
T1(W(Cab)) - - - - - - - - - - - - - - - >T2(R(Cab))

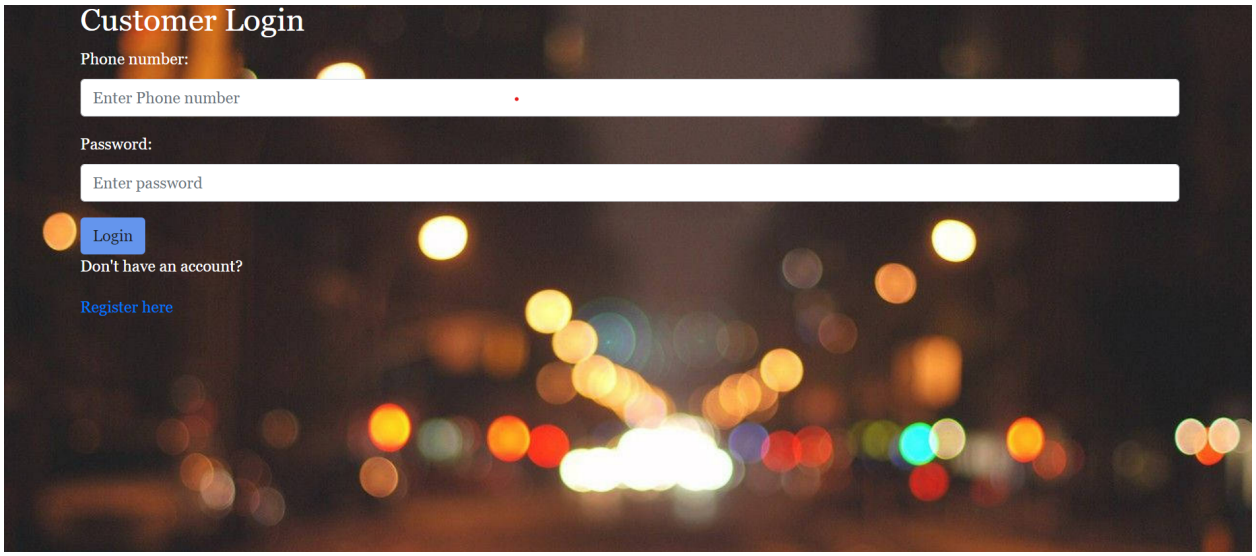
T1(W(Booking))<- - - - - - - - - - - - - - - - T2(W(Booking))

There is a cycle hence it is not conflict Serializable

Web application






Customer Login
Phone number:
Enter Phone number
Password:
Enter password
Login
Don't have an account?
Register here


Customer Dashboard
Welcome to the Customer dashboard.
Welcome n
Home
your bookings
Payment
Review
Logout
Pick up location
Pick up Location:
Enter the Location
Destination location:
Enter the Destination
submit