Cab Booking System
Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)

Project Scope:
Our project is based on a cab booking website similar to Uber. This would require creation of a backend system using Django and a MySQL database as well as a React.js frontend. The various functionalities would include cab booking information which would require the destination, number of passengers, handling of payments and complex features like real time tracking, estimated time of arrival and the drivers location.

Tech Stack:
1. Django
2. React.js
3. MySQL

Functional Requirements:
First, you will need to create the database structure, define the models and configure the application settings in Django. Additionally, you will need to create the frontend in React.js and use the Django Rest Framework to create APIs for accessing the backend. Finally, you will need to connect the frontend and the backend using the API and configure the application settings for the cab booking system.

Overall, the process requires knowledge of Python, Django, React.js, and MySQL. Additionally, you should have an understanding of API creation and other web technologies. With these skills, you should be able to develop a cab booking website similar to Uber.

If you are creating a cab booking website, you will need several different pages.

- The first page will be the homepage, which will include information about your cab booking service, your terms and conditions, and a call to action for users to sign up or log in.

- The second page will be the sign up/log in page, which will allow users to create an account or log in to an existing one.

- The third page will be the booking page, which will allow users to select their destination, the number of passengers, and other relevant details.

- The fourth page will be the payment page, which will allow users to select their payment method and enter their payment information.

- The fifth page will be the confirmation page, which will provide users with a confirmation of their booking.

- Finally, the sixth page will be the receipt page, which will provide users with a receipt of their booking.

All of these pages should be easy to use and intuitive for users to navigate. Additionally, all of the pages should be styled appropriately so that they match your brand's aesthetic.

For a cab booking website, a relational database management system (RDBMS) such as MySQL would be the best choice. MySQL is an open-source, powerful RDBMS that is used by many popular websites. It is reliable, easy to use, and can handle large amounts of data. Additionally, MySQL is compatible with many programming languages, including Python and Django, which are frequently used with web applications.

When using MySQL, it is important to define the database structure and the relationships between the data tables. Additionally, you should create indexes and stored procedures to optimize the database. Finally, you should use the appropriate data types for each column in the database to ensure that the data is stored correctly.

The database structure and relationships for a cab booking website should include the following tables:

- Users: This table will store the details of the users, including their name, email, password, and payment information.

- Bookings: This table will store the details of the bookings, including the origin and destination, the number of passengers, and the date and time of the booking.

- Drivers: This table will store the details of the drivers, including their name, email, and car details.

- Vehicles: This table will store the details of the vehicles that are available for hire, including the make, model, and registration number.

The relationships between these tables should be as follows:

- The Users table should be related to the Bookings table, as each user can have multiple bookings.

- The Bookings table should be related to the Drivers table, as each booking can only have one driver.

- The Drivers table should be related to the Vehicles table, as each driver can have multiple vehicles.

- By creating the appropriate relationships between these tables, the database will be able to store and retrieve the relevant data quickly and efficiently.

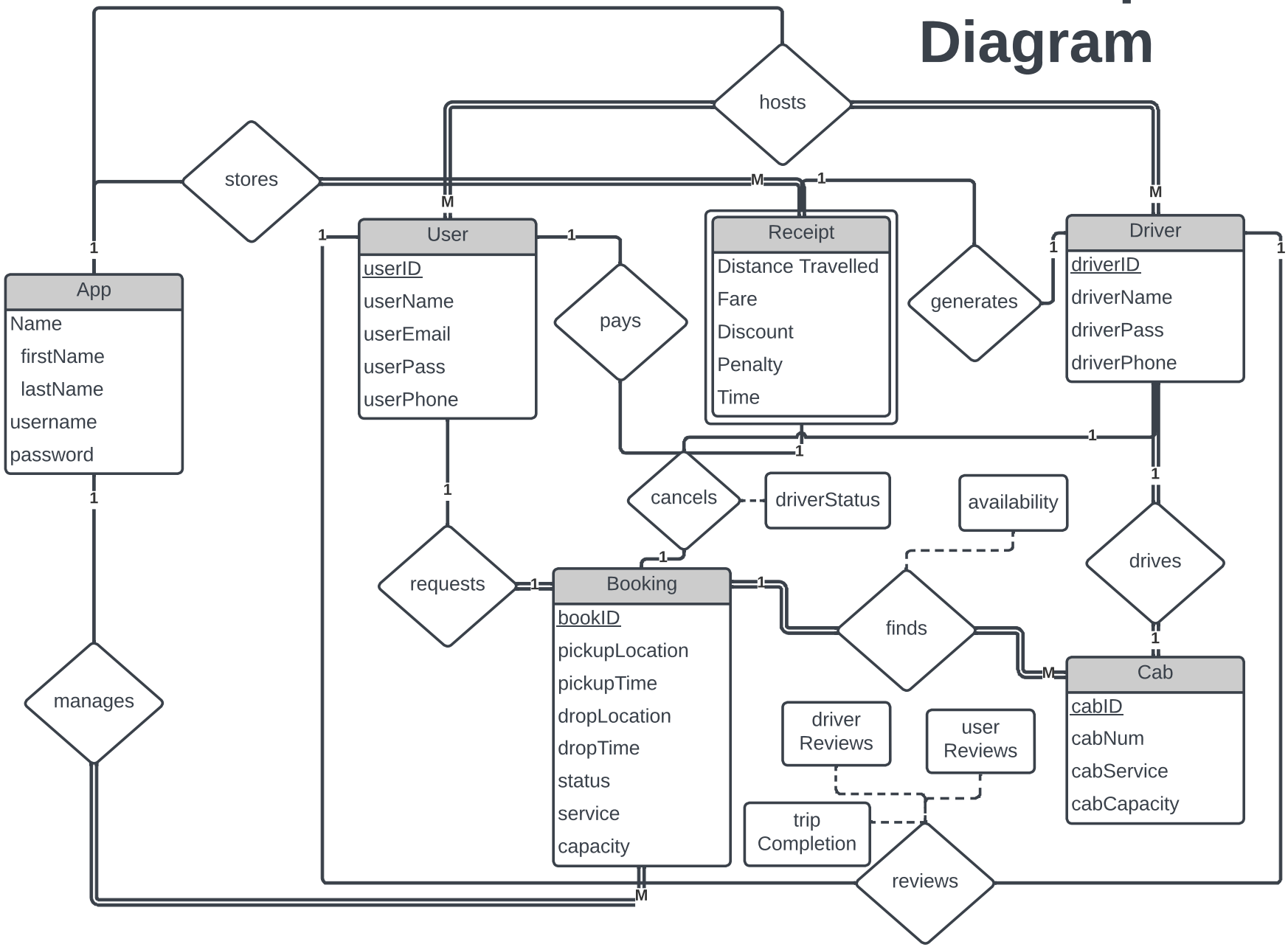
To make the cab booking website more complex, you can add additional features and functionalities. For example, you can add a real-time tracking system, so that users can see the driver's location and estimated time of arrival. You can also add an online payment system, so that users can pay for their rides in advance. Additionally, you can add a rating system, so that users can rate their drivers and provide feedback. Finally, you can add a rewards system, so that users can earn points and discounts for each ride they take. By adding these features and functionalities, you can make the cab booking website more complex and user-friendly.

In order to ensure a better app, there should be additional fields in the database to store more detailed data. For example, you can add a fare field to store the total fare for each booking. You can also add fields to store the ratings and reviews given by users, so that the app can show the average rating. Additionally, you can add fields to store the driver's profile photo and car photos, so that users can easily identify the driver. Finally, you can add fields to store the driver's route history, so that the app can provide better routes for future bookings. By adding these additional fields to the database, the app will be able to provide better services to its users.

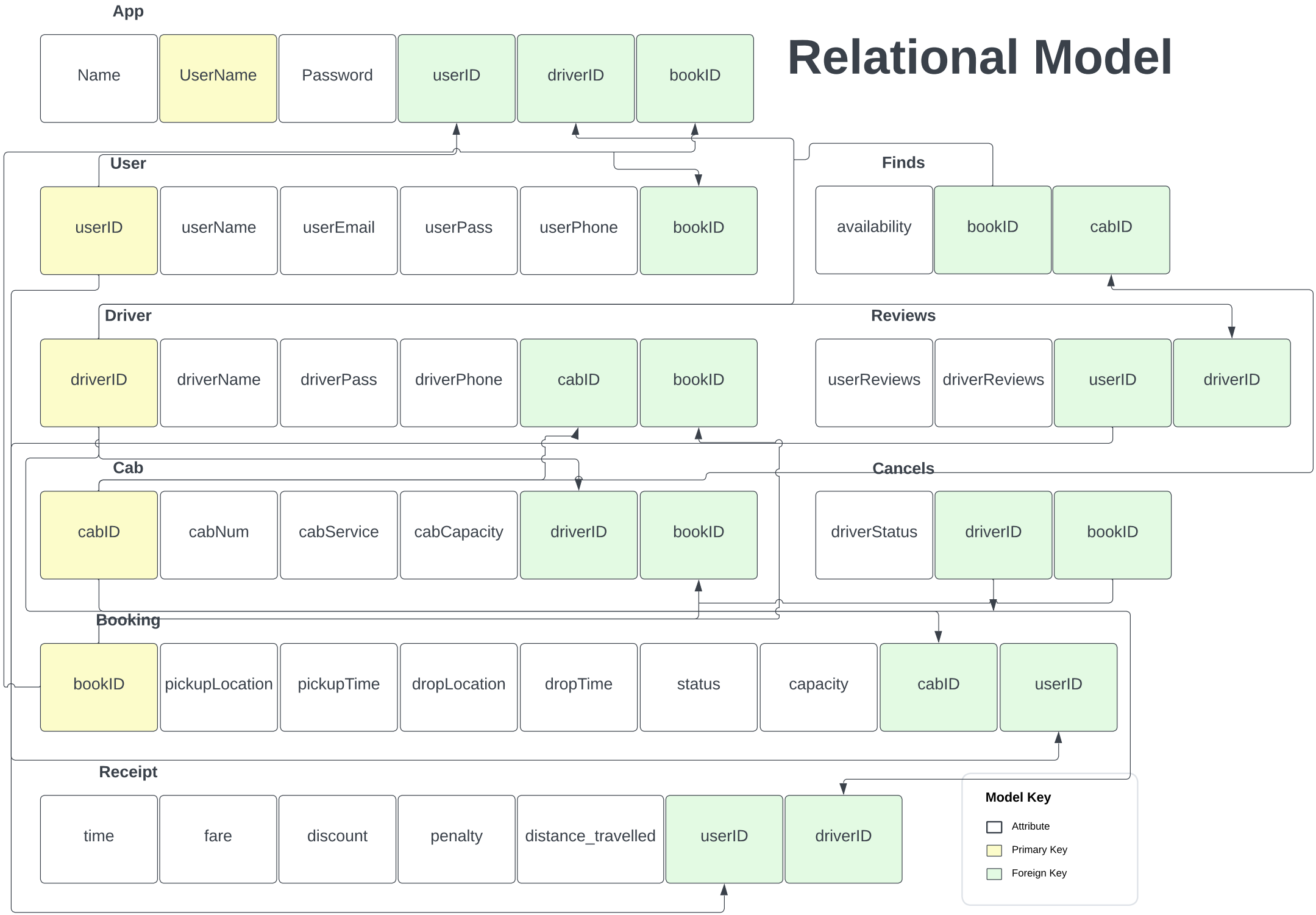
Entity Relationship Diagram

Parth Sarthi

DBMS CSE202 Project
Cab Management App



Relational Model



ParthSarthi | Cab Booking System | Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)

1. **App:**

   Attributes are name (firstName, lastName), username and password

   - Primary key is username
   - VARCHAR has been used for all the attributes
   - firstName and lastName concatenated
   - Foreign keys are userID (from user), driverID (from driver), bookID (from booking)

2. **User:**

   Attributes are userID, userName, userEmail, userPass and userPhone

   - Primary key is userID
   - INT used for userID, VARCHAR used for userName, userEmail, userPass and userPhone
   - Foreign key is bookID (from booking)

3. **Driver:**

   Attributes are driverID, driverName, driverPass and driverPhone

   - Primary key is driverID
   - INT used for driverID, VARCHAR used for driverName, driverPass and driverPhone
   - Foreign keys are cabID (from cab) and bookID (from booking)

4. **Cab:**

   Attributes are cabID, cabNum, cabService and cabCapacity

   - Primary key is cabID
   - INT used for cabID and cabCapacity and VARCHAR used for cabNum and cabService
   - Foreign keys are driverID (from driver) and bookID (from booking)

5. **Receipt:**
   Attributes are distanceTravelled, fare, discount, penalty and moment

   - Primary keys are userID, driverID and moment. These will be used as a discriminator
   - INT used for distanceTravelled, fare, discount, penalty and DATETIME for moment
   - Foreign keys are userID (from user) and driverID (from driver)

6. **Booking:**

   Attributes are bookID, pickupLocation, pickupTime, dropLocation, dropTime, currentStatus, service and capacity

   - Primary key is bookID
   - INT used for bookID and capacity, VARCHAR used for pickupLocation, pickupTime, dropLocation, dropTime, currentStatus and service
   - Foreign keys are userID (from user) and cabID (from cab)

7. **hosts:**

   Attributes are userID, driverID and username

   - INT used for userID and driverID, VARCHAR used for username
   - Foreign keys are userID (from user), driverID (from driver) and username (from app)

8. **Finds:**

   Attributes are availability, cabID and bookID

   - INT used for cabID and bookID, bool used for availability
   - Foreign keys are cabID (from cab) and bookID (from booking)

9. **Reviews:**

   Attributes are userReview, driverReview, tripCompletion, userID, driverID

   - VARCHAR used for userReview and driverReview, INT used for driverID and userID, bool for tripCompletion
   - Foreign keys are userID (from user) and driverID (from driver)

10. **Cancels:**

Attributes are driverID, bookID, driverStatus

- INT used for driverID and bookID, VARCHAR used for driverStatus
- Foreign keys are driverID (from driver) and bookID (from booking)

ParthSarthi | Cab Booking System | Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)

Queries:

1. Updating a specific value of an attribute:

```
UPDATE user SET userPhone='1234567891' WHERE userID=1;
SELECT * FROM user WHERE userID=1;
```

2. To check all the locations that the drivers of the company have been:

```
SELECT * FROM booking;
SELECT pickupLocation FROM booking
UNION
SELECT dropLocation FROM booking
ORDER BY pickupLocation;
```

3. To check if user has put same place as the pickup and drop location:

```
SELECT pickupLocation
FROM booking
INTERSECT
SELECT dropLocation
FROM booking;
```

4. Alter/Rename a table:

```
ALTER TABLE reviews
RENAME TO REVIEWS;
```

5. Adding a new user to the app:

```
INSERT INTO app (firstName, lastName, username, password)
VALUES ('Parth', 'Ganjoo', 'parth21342', 'ndcAbjs9');
SELECT * FROM app WHERE firstName='Parth';
```

6. Checking all the booked cabs with NATURAL JOIN:

```
SELECT * FROM cab NATURAL JOIN booking WHERE currentStatus='Booked';
```

7. Booking made for a specific cab:

```
SELECT * FROM cab WHERE cabNum='WBAXH5C55DD678413';
```

8. Popular drop locations in decreasing order:

```
select dropLocation, count(*) as popularDropLocations from booking
group by dropLocation
order by count(*) desc;
```

9. Payments done within a specific range of time:

```
SELECT * FROM receipt WHERE moment BETWEEN
'2023-01-01 00:00:00' AND '2023-02-17 23:59:00';
```

10. Number of Prime, XL, Eco and Sedan in inventory:

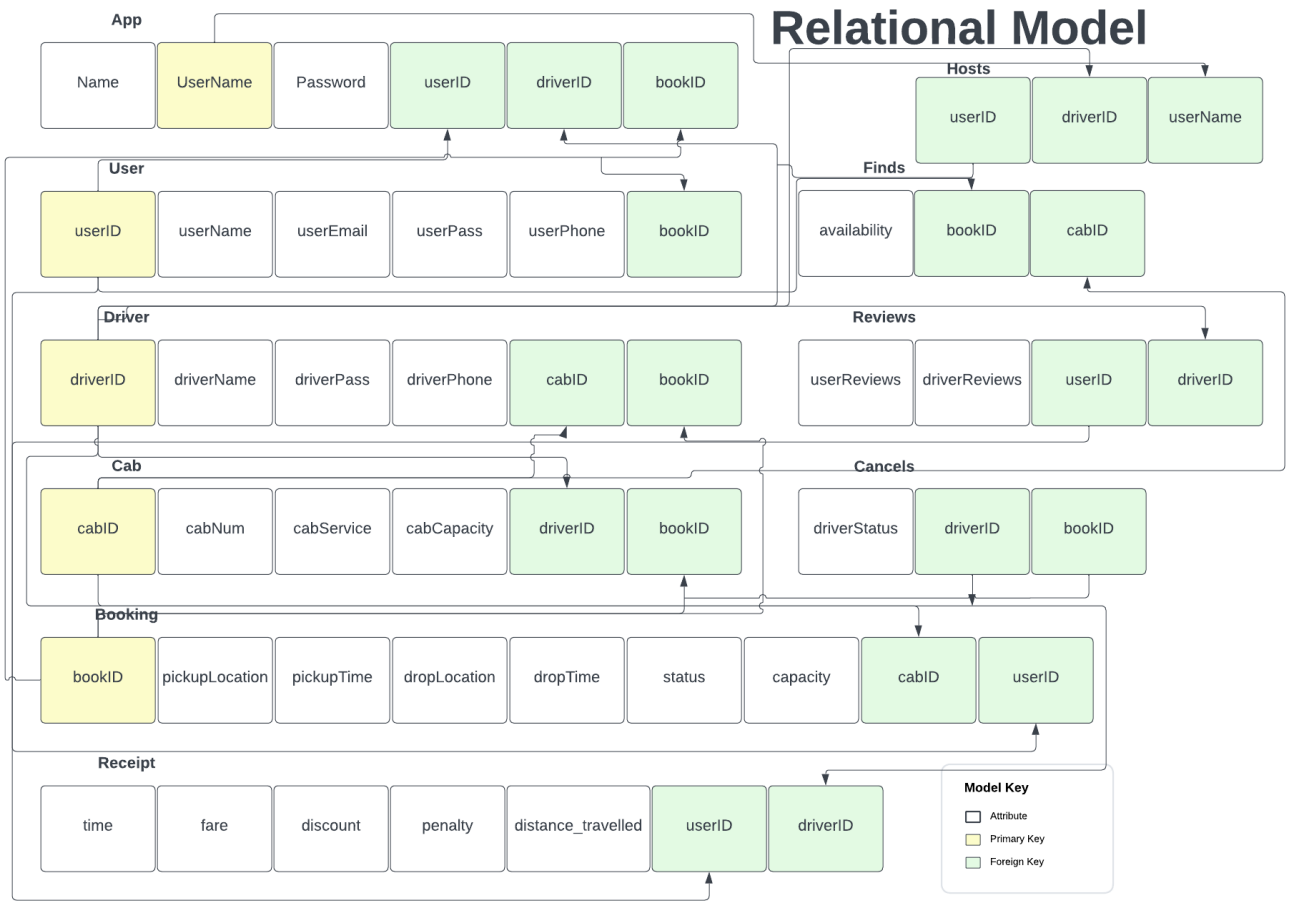
```
SELECT cabService, COUNT(cabService) AS cabServiceCount
FROM cab
GROUP BY cabService
ORDER BY COUNT(cabService) DESC;
```

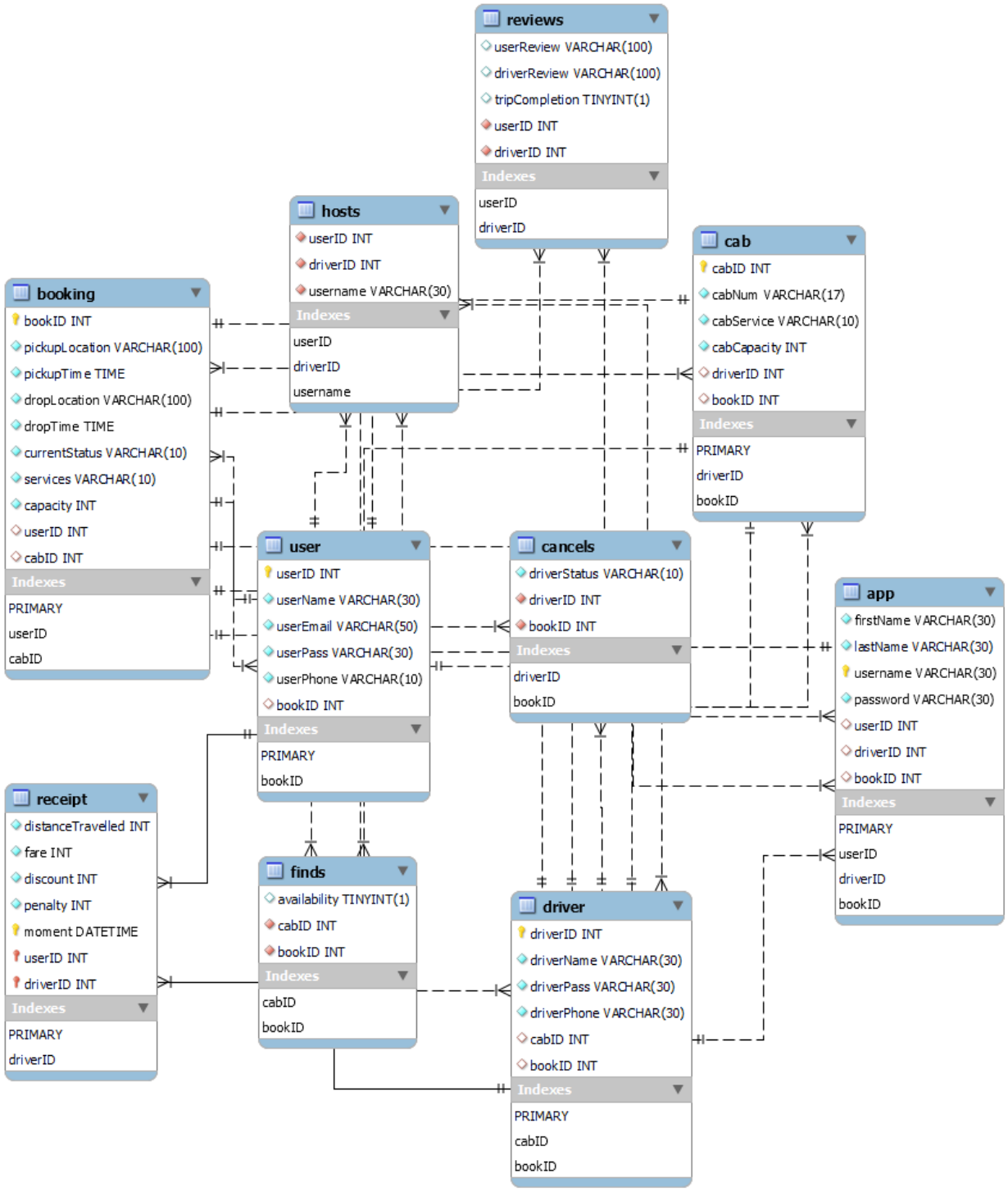
11. Selects all from booking and cab and checks capacity equal to 4:

```
SELECT *
FROM booking b
JOIN cab c ON b.cabID = c.cabID
WHERE c.cabCapacity = 4 AND b.capacity = 4
```

ParthSarthi | Cab Booking System | Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)




reviews
userReview VARCHAR(100)
driverReview VARCHAR(100)
tripCompletion TINYINT(1)
userID INT
driverID INT
Indexes
userID
driverID
hosts
userID INT
driverID INT
username VARCHAR(30)
Indexes
userID
driverID
username
cab
cabID INT
cabNum VARCHAR(17)
cabService VARCHAR(10)
cabCapacity INT
driverID INT
bookID INT
Indexes
PRIMARY
driverID
bookID
booking
bookID INT
pickupLocation VARCHAR(100)
pickupTime TIME
dropLocation VARCHAR(100)
dropTime TIME
currentStatus VARCHAR(10)
services VARCHAR(10)
capacity INT
userID INT
cabID INT
Indexes
PRIMARY
userID
cabID
user
userID INT
userName VARCHAR(30)
userEmail VARCHAR(50)
userPass VARCHAR(30)
userPhone VARCHAR(10)
bookID INT
Indexes
PRIMARY
bookID
cancels
driverStatus VARCHAR(10)
driverID INT
bookID INT
Indexes
driverID
bookID
app
firstName VARCHAR(30)
lastName VARCHAR(30)
username VARCHAR(30)
password VARCHAR(30)
userID INT
driverID INT
bookID INT
Indexes
PRIMARY
userID
driverID
bookID
receipt
distanceTravelled INT
fare INT
discount INT
penalty INT
moment DATETIME
userID INT
driverID INT
Indexes
PRIMARY
driverID
finds
availability TINYINT(1)
cabID INT
bookID INT
Indexes
cabID
bookID
driver
driverID INT
driverName VARCHAR(30)
driverPass VARCHAR(30)
driverPhone VARCHAR(30)
cabID INT
bookID INT
Indexes
PRIMARY
cabID
bookID

ParthSarthi | Cab Booking System | Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)

Embedded SQL queries:

OLAP queries:

- Check bookings of all users

  ```
  SELECT u.userID, u.userName, COUNT(b.bookID) as Total_Bookings
  FROM user u
  LEFT JOIN booking b ON u.userID = b.userID
  GROUP BY u.userID, u.userName WITH ROLLUP;
  ```

- The query will produce a result set with subtotals for each pickup location, as well as a grand total at the end.

  ```
  SELECT pickupLocation, COUNT(*) AS numBookings
  FROM booking
  GROUP BY pickupLocation WITH ROLLUP;
  ```

- Return the average distance traveled for each driver, as well as an additional row with the average distance traveled by all drivers combined

  ```
  SELECT driverName, AVG(distanceTravelled) AS average_distance
  FROM driver
  JOIN receipt ON driver.driverID = receipt.driverID
  GROUP BY driverName WITH ROLLUP;
  ```

- shows the cabs available for each type of service

  ```
  SELECT cabService, COUNT(*) as TotalCabs
  FROM cab
  WHERE cabID NOT IN (
     SELECT cabID
     FROM finds
     WHERE availability = false
  )
  ```

```
GROUP BY cabService WITH ROLLUP;
```

Triggers:

- Updates the distance travelled in the receipt

```
DELIMITER //
CREATE TRIGGER trg_receipt_distance_travelled
BEFORE INSERT ON receipt
FOR EACH ROW
BEGIN
   IF NEW.distanceTravelled <= 0 THEN
      SET NEW.distanceTravelled = 1;
   END IF;
END;
//DELIMITER ;
```

Test: INSERT into receipt (fare, discount, penalty, moment, userID, driverID, distanceTravelled) VALUES (100, 10, 0, '2008-11-11 13:23:44', 1, 1, -12);
SELECT * FROM receipt;

- Adds the current time and date into the receipt when booking is made

```
DELIMITER //
CREATE TRIGGER set_pickupTime_trigger
BEFORE INSERT ON booking
FOR EACH ROW
BEGIN
   SET NEW.pickupTime = NOW();
END;
//DELIMITER ;
```

ParthSarthi | Cab Booking System | Group 202

Parth Barthwal (2021341), Parth Ganjoo (2021342)

Transactions:
Non-conflicting:

1. Updating the penalty for a specific user ID:

   Start transaction;
   Update receipt set penalty = 100 where userID = 50;
   Commit;

2. Adding a new user to the app:

   Start transaction;
   Insert into app(firstName, lastName, userName, password)
   values('Parth', 'Ganjoo', 'parth21342', 'abcd1234');
   Commit;

3. Cancel all rides that are arriving, have arrived, or booked because of very bad weather/strike:

   Start transaction;
   Update cancels set driverStatus = 'Cancelled' where driverStatus = 'Arriving' or where driverStatus = 'Arrived' or where driverStatus = 'Booked';
   Commit;

Conflicting:

1. Locking receipt of userID 100 then updating the fare of userID 99. If another transaction is attempting to update the fare associated with userID 100 at the same time, it will be blocked until the lock is released:

   Start transaction;
   Select * from receipt where userID = 100 for update;
   Update receipt set fare = 200 where userID = 99;
   Commit;

2. One query locks username of a person for update and the other updates the phone number of the same person using their email:

   Select * from user where userName = ‘Merralee Middle’ for update;
   Update user set userPhone = ‘1234567890’ where userEmail = ‘mmiddle2r@umich.edu’;
   Commit;

User Interface code:

```
import MySQLdb
import random

booking=0
mydb1 = MySQLdb.connect(host = 'localhost',user = 'root',passwd = 'password',db = 'parthsarthi')
cur = mydb1.cursor()

def signup(usr,pswd):
    try:
        command = cur.execute(f"INSERT INTO user (userName,userPass,userID) VALUES ('{usr}','{pswd}','{usr}');")
        mydb1.commit()
        return "User creation success!"

    except Exception as e:
        print(f'Error: {e}')
        mydb1.rollback()
        return "User creation failed!"

def login(usr, pswd):
    try:
        command = cur.execute("SELECT * FROM user WHERE userName = '{usr}' AND userPass = '{pswd}';")
        result = cur.fetchone()
        if result:
            return "Login successful."
        else:
            return "Incorrect username or password."
```

```
    except Exception as e:
        print(f"An error occurred: {e}")
        return "Login failed."

def bookCab(usr):
    pickup=input("Enter Pickup Location: ")
    drop=input("Enter Drop Location: ")
    capacity=int(input("Enter Number of Passengers: "))
    try:
        command=cur.execute('SELECT cabID FROM cab WHERE cabCapacity >=
{capacity} AND cabService = "active";')
        result= cur.fetchone()
        if result:
            cabID=result[0]
            command1=cur.execute(f"INSERT INTO booking (userID, cabID, bookID,
pickupLocation, dropLocation, capacity) VALUES
('{usr}',{cabID},{booking},'{pickup}','{drop}',{capacity});")
            booking=booking+1
            command2=cur.execute(f'UPDATE cab SET cabService="booked" WHERE
cabID = {cabID};')
            command3=cur.execute(f'UPDATE cab SET bookID={booking} WHERE cabID =
{cabID};')
            command4=cur.execute(f'SELECT * cabNum FROM cab WHERE cabID =
{cabID};')
            cabNum=cur.fetchone()
            mydb1.commit()
            return f"Booking Successful with Cab Number: '{cabNum[0]}'"
        if not result:
            return "No available cabs."

    except Exception as e:
        print(f"An error occurred: {e}")
        mydb1.rollback()
        return "Failed to book the cab. Please try again later."

def completeTrip(usr):
    try:
        command=cur.execute(f'SELECT cabID FROM cab WHERE bookID = {booking}
AND status = "booked";')
        result=cur.fetchone()
```

```
        if result:
            cabID=result[0]
            tempFare=int(random.uniform(7.5, 75))
            command1=cur.execute(f'INSERT INTO receipt (userID, cabID, fare,
distanceTravelled) VALUES ("{usr}",{cabID}, {tempFare},{tempFare/2});')
            command2=cur.execute(f'UPDATE cab SET cabService="active" WHERE cabID
= {cabID};')
        else:
            return "No active bookings."
    except Exception as e:
        print(f"An error occurred: {e}")
        mydb1.rollback()
        return "Failed to complete the trip. Please try again later."

def welcome():
    print("Welcome to Parthsarthi")
    while(1):
        ch=int(input("New User? Sign Up(1)\nExisting User? Login(2)\nExit(3)\nEnter
Choice: "))
        if ch==1:
            usr=input("Enter Username: ")
            pswd=input("Enter Password: ")
            signup(usr,pswd)
        elif ch==2:
            usr=input("Enter Username: ")
            pswd=input("Enter Password: ")
            login(usr,pswd)
            console(usr)

        elif ch==3:
            print("Thank You for using Parthsarthi")
            exit(0)

        else:
            print("Invalid Choice, Retry")

def console(usr):
    print(f'Welcome {usr}!')
    inp=int(input("Book a New Cab(1)\nComplete Trip(2)\nExit(3)\nEnter Choice: "))
    if inp==1:
```

```
        bookCab(usr)
    elif inp==2:
        completeTrip(usr)
    else :
        print("Thank You for using Parthsarthi")
        exit(0)

welcome()
```

User guide:

1. Option to sign up and log in
2. Book a cab
3. Complete a trip