Final Project Report

Team 024 - DBMS Empire

1) Overall, our final project matched our project proposal. We achieved almost everything we wanted to achieve and kept as close as we could to our ground truth. The only point that we didn't implement was using a Google map API for our creative part, but we replaced it with another creative component which uses ML modeling to estimate price. Other than that, we made a few design changes in terms of schema which is described in section 3.

2) The application is fairly useful in fulfilling its purpose. Customers are able to search availability and specific cars to rent. On the flip side, lessors are able to list their cars on the web page. All core functionalities are covered. However, we could have made our webpage better and more user friendly. We don't report back to the user when there is a problem with the backend and having some functionality to alert them would be great. For example, renting a car that doesn't exist would throw nulls but no error message informing the user.

3) We did not change the source of our data as it fit our purpose well and was legitimate. However, we made a few tweaks to the UML diagram and the schema. Firstly, we realized FAQs do not really matter with any of the other entities. FAQs were connected to User before with the relation *'Asks',* however, we removed that and just put FAQs as a separate component with no relationships that solely serves to clear user's doubts to a certain extent - with pre-made questions and answers. We also removed the date of birth attribute of a user as it was not needed for the ideas of our app/component creation.

Secondly, the entity called ***rating*** in our original UML diagram was connected to the car entity ,but moving on with the development of our application, we decided to remove the relationship between ***car*** and ***rating*** ,as rating the car trip was sufficient enough rating for other users to make decisions on renting that car or not since the previous ratings were given by other users.

Thirdly, we added a new entity/table called ***carBookings*** that stores the current car bookings made by users. This entity has a primary key known as ***bookingID,*** along with the other attributes related to the user who booked the car. If a user books a car, it will disappear from the ***carAvailability*** table and get added into the ***carBookings*** table of the database.

4) With respect to the UML design, we did not change it and we implemented the exact structure as we proposed. But, with table implementations, we deleted the dateOfBirth column from the **User** table. In the initial proposal, we thought of checking for age validity using dateOfBirth but then we skipped the part. Also, a new user_id feature was added to the user table to uniquely identify the user. Moreover, the is_Leasor feature was converted from boolean to binary INT (0 or 1) for our convenience. But, we think that the older version of the tables (without user_id and boolean is_Leasure features) were better off compared to the new proposal as they yield more savings in memory explained in detail in part 9.

5) One functionality that we wanted to implement was a “Your Trip History” webpage, which shows all the past histories of the cars that a specific user rented before. However, because of time constraints, we decided not to implement the front-end part of that. We also wanted to connect a Google map API to our application to show where cars are located (instead of the barebones latitude and longitude system we have) but we found that it was too time-consuming to implement with the time that we had. Besides that, we implemented the features that we described in the proposal, including search, create, delete, update, rent, and top charts.

6) The advanced database programs complement our program in a way by providing constraints that were necessary in our application. We have decided to use the combination of stored procedure and trigger. The stored procedure takes care of two things. Firstly, it makes sure that a user does not book another car that he already has booked for the same timings. We solved this problem using a stored procedure as it iterates through the whole carTripHistory of that user and checks if he has a car booked at the same time. Secondly, the stored procedure also adds a free ride option for every 20th ride of the user by calculating the count of trip IDs in the user’s car trip history.

7)
Vivek Bhatt: I think a few things that our team faced were finding proper datasets that matched our needs. We didn’t find an exact dataset so we had to create data to fill in our database. Another big issue the team faced was with git and not making use of all of its functionality.
Sreekar Bathula: Furthermore, our team faced difficulties in routing the HTML pages as all the team members were new to full-stack development. We chose HTML/CSS for our front-end and Python Flask for our back-end. Not having much knowledge on these languages/tools, the team had to educate themselves before making any moves on the project. However, we were able to integrate everything successfully in a short amount of time!
Michael Trzupek: Another point that our group struggled with was getting everything setup/complete. It felt like the first 4 or weeks of the class were rather easy in terms of the project, but it picked up once stage3 was complete.
Praveen Kumar: Furthemore, the price prediction using machine learning with Python was simpler with all the feature engineering but it felt difficult to integrate the model to interact with the front and backend and project the results of the prediction back to the frontend. However, this was successfully completed on time with a lot of reference from internet search.

8) Regarding our initial ER/Diagram and tables for the project, we found that we had to actually make a new table in the backend to keep track of trip history for users. Furthermore, we found that we did not actually need a FAQ table in the backend, because that information can easily just be displayed on the frontend. We also decided not to store transactional data because of security concerns. Our user interface also looks quite different from what we initially drew up.

9) We could improve the application in many areas. We could implement the age verification part which was in our initial project proposal by simply comparing the dateOfBirth with legal

driving age. Moreover, additional features such as the make of the car, rating and frequency of the car use could also be added in the price prediction model for better and accurate results. We implemented a simple DecisionTreeRegressor algorithm but could also have tried other simple deep neural network algorithms. With respect to memory savings, the user_id feature could have been eliminated completely and email id could have been made as the primary key (as of our old design) to uniquely identify users. Also, is_Leasor feature could have been boolean instead of binary INT as boolean needs 1 byte of memory whereas binary INT may require up to 255 bytes.

10)

Praveen Kumar Murugaiah - Project proposal and initial database design, dataset search, data cleaning, database creation using MySQL and connecting the instance with Google cloud, Developing login, register pages, creating **search** functions (as part of CRUD) to search available cars, price prediction model and integration with HTML, CSS, JS and Python flask
Michael Trzupek - Final database design, Fullstack, **create** functions (as part of CRUD) in frontend & backend and connect them to HTML/CSS, Testing/Checking if data was entered correctly in the backend. The create functions include creating pages to add cars and car availability, trip history, part of car booking page and testing
Vivek Bhatt - Final database design, implementation of the **update** part (as part of CRUD), generation of missing feature dataset to suit our needs using Python, implementation of Triggers as part of advanced program & testing the connection with HTML/CSS and flask. The update part includes pages for the provision to edit the data in the car, user, carAvailability tables, part of car booking and testing.
Sreekar Bathula - Final database design,implementation of the **delete** part (as part of CRUD), review & rating data generation, implementation of Stored procedure as part of advanced program, testing the connection with HTML/CSS and flask. The delete part includes creating pages to delete car and user information.

STAGE 3 CORRECTION: FIXING FOREIGN KEYS

We fixed the foreign keys regarding the stage3 requirement for the SQL DDL commands.

```
CREATE TABLE Cars
(
  car_id VARCHAR (255) PRIMARY KEY,
  car_lat DOUBLE (3,10),
  car_long DOUBLE (3,10),
  make VARCHAR (255),
  model VARCHAR (255),
  year INT (4),
  VIN VARCHAR (255),
  owner_id VARCHAR (255) REFERENCES user(user_id)
);
```

```
CREATE TABLE carAvailability
(
     availability_id INT PRIMARY KEY,
     car_id VARCHAR (255) REFERENCES Cars(car_id),
     availability_from DATETIME,
     availability_till DATETIME
);

CREATE TABLE carTripHistory
(
     journey_id INT PRIMARY KEY,
     lesser_id VARCHAR(255),
     car_id VARCHAR(255) REFERENCES Cars(car_id),
     pickup_datetime DATETIME,
     dropoff_datetime DATETIME,
     price INT(4),
     rating INT
);

CREATE TABLE user
(
     user_id VARCHAR(255) PRIMARY KEY,
     email VARCHAR(255),
     password VARCHAR(255),
     is_leasor INT,
     firstName VARCHAR(255),
     lastName VARCHAR(255)
);

CREATE TABLE Ratings
(
     rating INT PRIMARY KEY,
     message VARCHAR(255)
);

CREATE TABLE carBookings
(
     carBookings ID INT PRIMARY KEY,
     firstName VARCHAR (255),
     lastName VARCHAR (255),
     lesserID VARCHAR (255) REFERENCES carTripHistory(lesser_id),
     pickup_datetime DATETIME,
     dropoff_datetime DATETIME
)
```