Middle East Technical University Northern Cyprus Campus

Computer Engineering Program

CNG352 Database Management Systems

METU NCC Transportation Management System
(METU NCC RideShare)

2246452 - Sameh Algharabli

2279073 - Aisan Sisani

2175263 - Elbaraa Elsaadany

Supervised by

Assoc. Prof. Dr. Yeliz Yesilada

Project title: METU NCC RideShare

Description of the application:

A web-based system that aims to assist METU NCC students with limited transportation in North Cyprus. For security purposes, students can login with their METU mail and a different password that the one used for login into METU NCC server. Each student will have a profile that shows their name, their contact information. Students can post a car ride event or a taxi ride event. Moreover, for reliability purposes, the system features the feedback/review functionality.

Car ride events enable students that own a car to share their ride with others or ask for others to pick them up. Taxi ride events allow students to share the fee of the taxi with others. Both the taxi ride and car ride events have some information such as date, time and location of departure and destination and also the number of students they can take with them.

The system enables students to see a list of events. the students can also filter the list of available events by any of the attributes mentioned for the ride events.

The system shall allow the students to request to join the ride events, these requests should also specify how many students will be with the student requesting for the join and also the contact information. Once, the request has been created the creator of the event can see the contact information of the requester and get in touch with them. The creator of the event, once contacted and decided, can accept the request and the information on the number of students able to join the event will be updated.

The events that are full and events which their departure time has passed will be deleted from the list of available events. Each student will be able to see private lists of events such as the list of events they have created, requested, gotten accepted to.

The system will store the information about the cars owned by the users. One car should be selected by the creator of a car event in which they are the drivers and also the drivers who agree to pick up other users should specify the car with which they will drive.

Additionally, the system will be able to store the details of taxi drivers. This data is just for the purpose of informing the users of possible taxi drivers and their numbers.

Data Requirements

METU Users

Our system's users are people such as students, staff, lecturer's who have a metu email. When signing up, the system will ask them to enter their information. The information or the data stored about each user are: METU *email, first name, last name, telephone number, whatsapp number, facebook* and *password*. The users will be able to create an event or request to join an existing event. Creating a user account is necessary in order to access the system. The rating attribute will also be stored for each user which is the average of all the ratings they have received.

Event ID, type (car or taxi), *price*, *date, time, departure location, destination location, number of seats*. Additionally we can store extra information about each event such as the *number of seats available* after each request has been accepted and also the *status* of the event which can be outdated, full, or available. Moreover, the creator of the event can add preferences to the events such as smoking, pets are allowed or not allowed.

Event

Events can be either a ride event or package delivery event. Moreover, each ride event can be a taxi *ride event* or *car ride event*. The taxi ride events are created when the user creating the event wants to share a taxi with other students. The car ride events are created when the creator of the event wants to drive by their car and wants to give a lift to other students **or** when a user is looking for a car owner to take them to their destination. In both cases . Therefore these information will be stored for each event: *Event ID, type* (car or taxi), *price*, *date, time, departure location, destination location*. The price for each event is set by the person who is responsible for the vehicle used for that event; in taxi ride events the person who calls the taxi is responsible and for car ride events the person who drives is responsible. Moreover, the creator of the event can add notes to the events such as smoking, pets are allowed or not allowed or any other detail.

Requests

Each user can request to join the events that are available (not closed or outdated). The data stored for each request include the *number of seats* that the requester requests for (this number should be less than the number of seats available for the ride event and for package events this number will be automatically zero), the *status* of the request, which can be either pending, rejected or accepted, and finally any *notes* which the requester wished to communicate with the creator. The communication between the metu users will be possible through contact information such as telephone numbers.

Reviews

To make this application more reliable for the users, the creator of the ride events can create a review for each requester whose request has been accepted. Also the requesters whose request has been accepted can leave a review for the creator of the event. Thus, the data stored for each review include the *time* of the review, *rating* (out of 5), *comments.*

Vehicle

For the safety and trustiness purposes, the system does store the information of the Vehicles. A vehicle can be either a taxi or a private car. This information will be stored by both types of vehicles: unique vehicle ID, vehicle model, licence plate no, car's capacity. Each vehicle can only be registered to one person, either a taxi driver or a user.

Borrowing Cars

The system will enable the users to borrow cars from other users. The owner will be able to reject the requests to borrow a car.

Taxi Drivers

The following information should be stored for each registered taxi driver: name, surname, telephone number, city of residence licence number, identification number, type of identification(passport/TC ID), company name (optional), email and password. A taxi driver must have a car to register to the system. The taxi driver is not a typical user and cannot create or request for events, the only activities done by taxi drivers are requesting to accept a taxi reservation and logging in.

Taxi Reservation

The metu users are able to create taxi reservations which indicates the time and departure and destination addresses. The taxi drivers will be able to see these reservations and request to drive these taxi reservations. The taxi driver should also specify the price of the ride. then the metu user will be able to see the requests made for each reservation that they made and accept the driver. The communication between both parties will be done by their contact information such as telephone number.

Package delivery

If a user wants to deliver a package somewhere, they can make a package delivery event where they look for a car owner to do the delivery for a price. The weight and content of the package will be stored.

Transaction Requirements

Data Entry

- Enter the details of a new user.
- Enter the details of a new event.
- Enter the details of a Review.
- Enter the details of a Request.
- Enter the details of a new vehicle.
- Enter the details of a new taxi driver.
- Enter the details of a new taxi reservation.
- Enter the details of a new taxi reservation request.

Data update/delete

- Update/Delete the details of a user.
- Update/Delete the details of an event
- Update/Delete the details of a request.
- Update/Delete the details of a Vehicle.
- Update/Delete the details of a Taxi Driver.
- Update/Delete the details of a taxi reservation
- Update/Delete the details of a taxi reservation request

Data Queries

1. List all the users which their id and password is equal to the given values
2. List all the details of a user based on their id.
3. List all the events created by a specific user based on their id
4. List all the users who joined a specific event.
5. List all the reviews made from a specific event.
6. List all the reviews made from a specific event for a specific user.

7. List all the events requested by a specific user based on their id
8. List all the upcoming (available or full) events sorted by date and time.
9. List all the available events.
10. List all the events based on the specific attributes (search filters such as price, date, number of seats, ...).
11. List details of all the users ordered alphabetically except their password
12. List all the users who has an available event
13. List all the reviews made by a user.
14. List all the reviews given to a user.
15. list all the reviews associated with a car.
16. List all the users whose request has been accepted for a specific event.
17. List all the users who have requested to join a specific event.
18. List all the cars.
19. List all the cars owned by a user.
20. List all the borrow requests for a specific car.
21. List all the taxi drivers.
22. List all taxi reservations.
23. List all taxi reservations requests.
24. List taxi reservations made by a specific user.
25. List all the taxi reservations whose status is driver_found.
26. List all the taxi reservations whose status is driver_found created by a specific user.
27. List all the package deliveries.
28. List all the package deliveries made by a specific user.
29. List all the package deliveries delivered by a specific user.
30. List all events that are package delivery type.

EER Assumptions

1. The user can create an event to send a package, drive people or ask for a ride.
2. The user "requests the event" means that he offers to deliver the package, offers to drive people or requests to join a ride.
3. An event can either be a package delivery event or a ride event. However, If a driver accepts a ride event, he can also accept a package event and vice versa(Note: they are different events).
4. Only the one who wants to send a package can create a package delivery event.
5. Package is not delivered by taxi, because the taxi driver is not a metu user of the system.
6. Ride_type is an attribute that specifies the type of the ride whether it is taxi ride or a car ride.

7. Responsible_for_vehicle is a relationship which decides who is responsible for the vehicle of the event. However there are some exceptions and rules:
    a. The creator of a package event cannot be the person responsible for the event. The person who requests for the package and is accepted is the driver for the package.
    b. The person responsible for the vehicle in the case of ride events can be either the creator or the person who has requested to join an event and got accepted.
    c. The creator of the taxi events must be the person responsible for the vehicle. We should also note that the person responsible for the taxi events are not actual drivers but only they are the person who will manage a taxi and call the taxi driver.

    Note: the person who does the driving does not necessarily have a car, but can borrow a car. In this case the system first checks if the person who is requesting or creating a event and wants to be responsible for the ride events are actually having a car or has borrowed a car in the time the event will happen

8. Rating in Metu_user is an attribute that stores the overall rating of a metu_user.
9. The no_reviews attribute of the METU User entity will store the number of reviews given (not created) to the metu user. The purpose of this attribute is to calculate the new rating of the user much faster in case of an insertion of a new review given to this user. (instead of counting the number of reviews given to the user everytime a review is created for the user)
10. Req_status concerns the participants of the event, whether their request is accepted or not. While the status in the event concerns the status of the event itself, whether the event is active or canceled.
11. The responsible person for the taxi ride is the creator of the taxi ride.

Mapping:

1-Strong entities and Generalization:

- Metu_User(id, metu_mail,facebook, no_reviews,rating,password,Whatsapp,firstname,surname,phone)

- Event(<u>Eid</u>, event_status, data, time, price, notes, destination, departure)

- Vehicle(<u>vehicle_id</u>, vehicle_model, vehicle_capacity, licence_plate_no, vehicle_color,)

- Taxi_Driver(mail, licence number, name, surname, password, phone, whatsapp, facebook, company name, city)

- Taxi(vehicle_id[FK:Vehicle.vehicle_id])

- Package(EID[FK:Event.Eid],weight,content)

- Ride(EID[FK:Event.Eid], total_seats,ride_type)

- Car(vehicle_id[FK:Vehicle.vehicle_id])

- Taxi_Reservation(Id,time,location,destination, date, status)

2- Weak entities:

Review(user_to_id[FK:METU_User:id], user_from_id[FK:METU_User:id], eventId[FK:Event.Eid], time, comment, rating)

3- 1:1 relationship:

Owns relation(Foreign key approach):

- Car(vehicle_id[FK:Vehicle.vehicle_id], license_number, user_from[FK:METU_User:id])

Owned by relation(we combined the taxi and taxi_driver):

- Taxi(vehicle_id[FK:Vehicle.vehicle_id], mail, licence number, name, surname, password, phone, whatsapp, facebook, company name, city)

4- 1:M relationship:

Creates relation:

- Event(Eid, event_status, data, time, price, notes, destination, departure,creator_id[FK:METU_User:id])

Delivered_by relation:

- Package(EID[FK:Event.Eid],weight,content,car_id[FK:Car.vehicle_id])

Responsible_for_vehicle relation:

- Event(Eid, event_status, data, time, price, notes, destination, departure, reponsible_user_id[FK:METU_User:id])

Driven_by relation:

- Ride(EID[FK:Event.Eid], total_seats, ride_type, vehicle_id[FK: Vehicle: vehicle_id],reponsible_user_id[FK:METU_User:id])

Creates_reservation relation:

- Taxi_Reservation(Id,time,location, destination, date, status, creator_id[FK: METU_User:id])

5- M:M relationship:

Request relation:

Request(user_from[FK:METU_User:id], EID[FK:Event.Eid], notes, time_stamp, req_status)

Reservation_Request relation:

Reservation_Request(taxi_id[FK:Taxi.vehicle_id], reservation_id[FK:Taxi_Reservation.id], price, status)

Borrow relation:

Borrow(user_from[FK:METU_User:id], car_id[FK:Car.vehicle_id], status, from_time, to_time)

6- Last Version:

- Metu_User(id, metu_mail,facebook, no_reviews,rating,password,Whatsapp,firstname,surname,phone)

- Event(<u>Eid</u>, event_status, data, time, price, notes, destination, departure, creator_id[FK:METU_User:id], reponsible_user_id[FK:METU_User:id])

- Package(<u>EID</u>[FK:Event.Eid],weight,content,car_id[FK:Car.vehicle_id])

- Ride(<u>EID</u>[FK:Event.Eid], total_seats, ride_type, vehicle_id[FK: Vehicle: vehicle_id])

- Vehicle(<u>vehicle_id</u>, vehicle_model, vehicle_capacity, licence_plate_no, vehicle_color)

- Taxi(<u>vehicle_id</u>[FK:Vehicle.vehicle_id], mail, licence number, name, surname, password, phone, whatsapp, facebook, company name, city)

- Car(<u>vehicle_id</u>[FK:Vehicle.vehicle_id], license_number, user_from[FK:METU_User:id])

- Taxi_Reservation(<u>Id</u>,time,location, destination, date, status, creator_id[FK: METU_User:id])

- Review(<u>user_to_id</u>[FK:METU_User:id], <u>user_from_id</u>[FK:METU_User:id], <u>eventId</u>[FK:Event.Eid], time, comment, rating)

- Request(<u>user_from</u>[FK:METU_User:id], <u>EID</u>[FK:Event.Eid], notes, time_stamp, req_status)

-Reservation_Request(<u>taxi_id[FK:Taxi.vehicle_id], reservation_id[FK:Taxi_Reservation.id]</u>, price, status)

-Borrow(<u>user_from</u>[FK:METU_User:id], <u>car_id</u>[FK:Car.vehicle_id], status, from_time, to_time)

Mapping Assumptions

- We have decided to merge taxi and taxi drivers under a taxi entity, because they have a 1:1 relation and both of them are in total participation. Therefore, a taxi driver can only register with one car.

Normalization

FDs:

The complete explanation and rules of each normal form is written below and just once and the small details are written after each relation.

1NF: a relation is in 1NF if:

1. This relation does not contain any composite or multi-valued attribute, meaning that all the attributes are single valued attributes.

2. The attributes domain does not change later on in the software.
3. Each attribute has a unique name.

2NF: a relation is in 2NF if:

1. It is already in 1NF
2. There is no partial functional dependency, that is no non-prime attribute should be functionally dependent on the prime attribute.

3NF: a relation is in 3NF if:

1. It is already in 1NF
2. It is already in 2NF
3. There are no transient FDs meaning that every non prime attribute is non-transitively dependent on superkey. Also referring to the formal definition of 3NF we can see that LHS of all the FDs are superkeys.

   Formal definition: A relation R having functional dependency A–>B is in 3NF if either of the conditions given below are true .

   a. A is a superkey.
   b. B is the prime attribute, that is B is the part of the candidate key.

BCNF: a relation is in BCNF if:

It is already in 1NF

It is already in 2NF

It is already in 3NF

For every functional dependency X → Y, X is the super key of the table.

METU_User:

id→ Metu_mail , facebook, no_reviews,rating,password,Whatsapp,firstname,surname,phone

Metu_mail → id, facebook, no_reviews,rating,password,Whatsapp,firstname,surname,phone

Facebook → id, Metu_mail, no_reviews,rating,password,Whatsapp,firstname,surname,phone

Whatsapp → id, Metu_mail,facebook, no_reviews,rating,password,firstname,surname,phone

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since Whatsapp, facebook and id are all candidate keys.

Event:

Eid → event_status, data, time, price, notes, destination, departure, creator_id[FK:METU_User:id], reponsible_user_id[FK:METU_User:id])

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since Eid is the primary key.

Vehicle:

Vehicle_id → vehicle_model, vehicle_capacity, licence_plate_no, vehicle_color

Licence_plate_no → Vehicle_id , vehicle_model, vehicle_capacity, vehicle_color

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since Vehicle_id and Licence_plate_no are all candidate keys.

Taxi:

Vehicle_id → mail, licence number, name, surname, password, phone, whatsapp, facebook, company name, city

mail → vehicle_id, licence number, name, surname, password, phone, whatsapp, facebook, company name, city

facebook → vehicle_id,mail, licence number, name, surname, password, phone, whatsapp, company name, city

licence number → vehicle_id, mail , name, surname, password, phone, whatsapp, facebook, company name, city

whatsapp → vehicle_id,mail, licence number, name, surname, password, phone, facebook, company name, city

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since Vehicle_id, mail, facebook, licence number and Whatsapp are all candidate keys.

Package:

EID → weight, content, car_id

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS EID is the primary key.

Ride:

EID → total_seats, ride_type, vehicle_id

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS EID is the primary key.

Car:

Vehicle_id → license_number, user_from

License_number → Vehicle_id ,user_from

User_from → license_number,Vehicle_id

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since vehicle_id and license_number and user_from are all candidate keys.

Taxi_reservation:

Id → time,location, destination, date, status, creator_id

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF the primary key is not composite and also regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS Id is the primary key

Request:

user_from, EID → notes, time_stamp, req_status

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF all the non prime attributes are dependent on both of the prime attributes in the composite primary key. Regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS user_from, EID is the composite key

Review:

user_to_id, user_from_id, eventId → time, comment, rating

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF all the non prime attributes are dependent on all of the prime attributes in the composite primary key. Regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS user_to_id, user_from_id and eventId is the composite key.

Reservation_request:

taxi_id, reservation_id → price, status

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF all the non prime attributes are dependent on all of the prime attributes in the composite primary key. Regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS taxi_id and reservation_id is the composite key.

Borrow:

user_from, car_id → status, from_time, to_time

This relation is in 1NF, 2NF, 3NF and BCNF. Regarding 2NF all the non prime attributes are dependent on all of the prime attributes in the composite primary key. Regarding the 3NF and BCNF the LHS of all the FDs are superkeys since the only LHS user_from and car_id is the composite key.

Denormalization and Last Modification

For the denormalization we have combined the taxi driver and the taxi relation together and called the combined relation taxi since the relation was one to one with total participation on each side. We don't need any other refinements specially considering the cardinalities of the relation as they are not a big number. Thus denormalizing more will not have dramatic influence over the speed of the queries and rather will increase the storage size which will be a negative change. Also, the attributes names and the relation are clear. Thus the last version of the database seems suitable for our application.

Analysis

- Metu_Users(<u>id</u>, metu_mail, facebook, no_review, rating, password, Whatsapp, firstname, surname, phone)

There are around 3000 metu students on campus on average and a maximum of 300 metu staff. We estimate that around one third of these individuals will register to the system. Thus, the number of rows in the METU_USERS table will be around 1100. Thus the number is an average on the number records each year. As older students get graduated the system will allow the students to delete their accounts, also deletes the deprecated accounts annually based on the graduation, teh registration date and also the activity of the user.

We expect rare updates on all the columns except the rating and the no_reviews since it will be updated based on the reviews made for the user. When updates such as no. 3 happens then the id of the users will be used, thus we have added an id column to be an int primary key so that the indexing would be more effective as metu mail is a string with at least 2 chars. However still the metu_mail will be a unique alternative key and will be used for logging into the system. Thus we need the index for it. However by declaring it UNIQUE in th MySQL the innoDB engine automatically creates an index for this column as it is unique.

Also besides the update queries the most of the select queries will have a join with the metu_users table, thus choosing a small primary that has an index is the best practice in this case, specially since the table will be ordered based on the primary key and the primary key columns will be stored besides the other columns specified in the secondary index (index for the metu_mail) in the secondary index rows (this primary key will then be used for row lookup in the actual table).

Also adding the index for id and metu_mail will not be a burden from the aspect of updating or deleting or inserting records, since the id and metu_mail will never change and we would not have daily deletes on the users but rather annually. Also after most of the users have registered to the system each year the system in the later months of the year there would be minimal inserts to the system.

Summary:

Primary index for id

Secondary index for metu_mail

Not nulls: id, metu_mail, first_name, surname, whatsapp, no_review, password

Primary key: id

Alternative key / unique: metu_mail

Foreign keys: non

Domain constraints: no_review >= 0, rating >= 0

Default values: no_reviews = 0

- Car(<u>vehicle_id</u>[FK:Vehicle.vehicle_id], license_number, user_from[FK:METU_User:id])

We are estimating that 80 percent of the metu staff have a car. And around only 10 percent of the students have cars. Also, we are estimating that each user will only have one car on average. Thus, by referring to the estimated number of records in the METU_USERS we can estimate that around 180 cars will be registered to the system. As the number of metu users will not change on average each year thus the number of records in this system will not change as well.

We expect barely any updates on car records since the vehicle id doesn't change and the owner might change rarely. Most of the select operations will be related to the user_from since the system wants to find the cars owned by a user such as select 5 and 6 as when the user wants to create or request to drive an event. Also we expect low frequency for the delete and insert queries. Thus we can have a secondary index on the user_from column that will speed up the select and will not affect the insert, delete or update.

Summary:

Primary index for vehicle_id

Secondary index for user_from

Not nulls: vehicel_id, license_number, user_from

Primary key: vehicle_id

Alternative key / unique: non

Foreign keys: vehicle_id[FK:Vehicle.vehicle_id], user_from[FK:METU_User:id]

Domain constraints: non

Default values: non

- Taxi(vehicle_id[FK:Vehicle.vehicle_id], mail, licence number, name, surname, password, phone, whatsapp, facebook, company name, city)

North Cyprus is relatively a small island. We can estimate the number of taxis in each area from the following table.

area	**Number of companies**	**Number of taxis per company**	**Total taxies**	**Total taxies registered**
lefke	**2**	**5**	**10**	**5**
guzelyurt	**3**	**5**	**15**	**7**
kyrenia	**30**	**5**	**150**	**7**
nicosia	**10**	**7**	**70**	**3**
famagusta	**20**	**5**	**100**	**5**
totall				**27**

We expect that 50% of taxis in the closer areas to the campus such as guzelyurt and lefke will be registered to the system and only 5% for other areas. Thus we can estimate at most 30 taxies will be registered to the system. We estimate that the number of records will not change dramatically after most of the taxies in the TRNC are registered to the system.

The mail is an alternative key for the taxi and it can be used for login. Thus there will be a secondary index assigned to it automatically since it will be UNIQUE.

As the number of records will be around 30 and the predicates of queries are related to the vehicle_id such as select 3, there will be no need for any index. Updates, deletes and insert are very rare as well.

Summary:

Primary index for vehicle_id

Secondary index for mail

Not nulls: all except company_name, city and facebook

Primary key: vehicle_id

Alternative key / unique: mail

Foreign keys: vehicle_id[FK:Vehicle.vehicle_id]

Domain constraints: non

Default values: non

- Vehicle(<u>vehicle_id</u>, vehicle_model, vehicle_capacity, licence_plate_no, vehicle_color)

The number of records in this table is the addition of records in the car and taxi tables thus the estimated number of records in this table is 180+30 = 210.

Licence_plate_no is UNIQUE thus an automatic secondary key will be created. As the number of records will be around 30 and the predicates of queries are related to the vehicle_id, there will be no need for any index. Updates, deletes and insert are very rare as well.

Since the select and join operations' predicate is related to vehicle_id the only index needed is the primary index. There will be very rare updates and deletions on this table also the insert will be around 210 as cars are registered. Since the

Summary:

Primary index for vehicle_id

Secondary index fro Licence_plate_no

Not nulls: all

Primary key: vehicle_id

Alternative key / unique: Licence_plate_no

Foreign keys: vehicle_id[FK:Vehicle.vehicle_id]

Domain constraints: (vehicle_capacity >= 1)

Default values: non

-Borrow(<u>user_from</u>[FK:METU_User:id], <u>car_id</u>[FK:Car.vehicle_id], status, from_time, to_time)

We don't expect this functionality of the system to be used rarely. We expect the number of inserts in this table to be at maximum around 3 cars each week. Thus each year around 130 borrows will be recorded (52 weeks in a year). As the deprecated borrows will be deleted annually this will be the same each year. Since this feature is used rarely and not many records we don't need any index other than the primary key index that will be created automatically.

Summary:

Primary index for user_from, car_id

Secondary index: non

Not nulls: all attributes

Primary key: user_from, car_id

Alternative key / unique: non

Foreign keys: **<u>user_from</u>[FK:METU_User:id], <u>car_id</u>[FK:Car.vehicle_id]**

Domain constraints: (status IN ('Pending', 'Accepted', 'Rejected', 'Canceled')),

Default values: status: ‘pending’

- Event(<u>event_id</u>, event_status, datetime, price, notes, destination, departure, creator_id[FK:METU_User:id], reponsible_user_id[FK:METU_User:id])

We expect that this table will be used the most by users. Since this is the main functionality of the system. To estimate the number records that will be stored in this table we have to estimate the number of packages and the ride events, by referring to this number is: 250 + 1800 = 2050.

We expect rare updates on the columns such as data, time, price, notes, destination, departure. However we expect changes more often on the event_status of the event as it can change between ‘Active’ to ‘Full’ or to ‘Canceled’. And very few updates on the responsible_user_id. In update to the responsible_user_id and to event_status the update will be done one at a time to the rows with specific id (**<u>event_id</u>**). Thus selecting the Eid as the primary key which will have an automatic PRIMARY index is the best practice in updates that include select.

Most of the select queries will have a predicate related to the event_status and the datetime. The Users will want to see the events that are still active and also the date has not been passed. Thus we expect a lot of the queries to be involved with these types of queries such as select queries of 11 and 12. Thus, secondary indexes for the datetime and event_status will be created for this table. Although we should know that we can have select queries compatible with the search filters that users select such as the destination, departure and price such as 10, creating an index will not help to optimize the query. As these filters will not be used frequently by each user and also the number of records which is 2050 is not a very big number.

Although creating the index on event_status may cause the updates to be slower, the number of select queries on this table that involve this attribute is much more than the insert or update or delete operation. Same is true for the datetime.

Summary:

Primary index for event_id

Secondary index for event_status and datetime

Not nulls: event_id, datetime, event_status, price, destination, departure, creator_id

Primary key: event_id

Alternative key / unique: non

Foreign keys: creator_id(metu_users(id)), responsible_user_id(metu_users(id))

Domain constraints: price>= 0, (`event_status` IN ('Active', 'Full', 'Canceled'))

Default values: event_status= ‘Active’

- Package(<u>EID</u>[FK:Event.Eid],weight,content,car_id[FK:Car.vehicle_id])

This event will be used less compared to the ride event. The number of created packages each week will be no more than 5. Thus each year 250 package events will be created.

We expect a rare update and delete on the package. The most of the select queries will be related to the EID and as the number of these events will be very small (250). There is no need for any other index other than the primary index for the primary of EID.

Summary:

Primary index for EID

Secondary index : non

Not nulls: eid, weight, content

Primary key: EID

Alternative key / unique: non

Foreign keys: (eid) REFERENCES event(event_id) and (car_id) REFERENCES car(vehicle_id)

Domain constraints: non

Default values: non

- Ride(<u>EID</u>[FK:Event.Eid], total_seats, ride_type, vehicle_id[FK: Vehicle: vehicle_id])

We expect that on weekdays around 5 ride events will be created and on weekends this number will be doubled to 10 events. Thus, yearly 1800 ride events will be created ((10*2 + 5*5) * 50 weeks = 1820).

Regarding the select queries most of them will be related to the primary key, and the ride_type because we expect that people want to search for either a car ride or a taxi ride thus creating a secondary index on ride_type will optimize these search queries. We expect updates on the vehicle_id and the total_seats not on the ride_type and the delete will be very rare. As for the insert since the data is not that huge we can make a tradeoff by creating an index for the ride_type since the frequency of the select queries will be more than insert.

Summary:

Primary index for EID

Secondary index : ride_type

Not nulls: eid, ride_type

Primary key: EID

Alternative key / unique: non

Foreign keys: vehicle_id[FK: Vehicle: vehicle_id]

Domain constraints: (total_seats >= 1)

Default values: non

- Request(<u>user_from</u>[FK:METU_User:id], <u>EID</u>[FK:Event.Eid], notes, time_stamp, req_status)

We expect that for each package on average 2 requests will be created and for ride events 5 requests. Thus the number of request records will be around 10000 (2*250 + 5 * 1800 = 9500) annually.

Regarding the select queries all of them will be related to the primary key because there are no other unique attributes to be made a secondary index. We expect updates on the EID and the delete will be very rare.

Summary:

Primary index for user_from , EID

Secondary index : none

Not nulls: All except notes

Primary key: user_from , EID

Alternative key / unique: non

Foreign keys: user_from[FK:METU_User:id], EID[FK:Event.Eid]

Domain constraints: req_status IN (‘Pending’, ‘Accepted’, ‘Rejected’)

Default values: req_status=’Pending’

- Review(<u>user_to_id</u>[FK:METU_User:id], <u>user_from_id</u>[FK:METU_User:id], <u>eventId</u>[FK:Event.Eid], time, comment, rating)

The number of participants in a package event is 2 and the number of participants in a ride event on average is 3. If we assume that 50% of the participants will create a review therefore the number of records will be around 3000 reviews (250 * 2 *50/100 + 1800*3*50/100 = 2950) annually.

Regarding the select queries, most of them will be related to the primary key, because we expect that people want to search for the reviews of a specific user such as the select no 8. We expect rare updates and deletes to be done on the review table.

Summary:

Primary index for (user_to_id, user_from_id, eventId)

Secondary index : non

Not nulls: all the columns except comment

Primary key: user_to_id, user_from_id, eventId

Alternative key / unique: non

Foreign keys: user_to_id[FK:METU_User:id], user_from_id[FK:METU_User:id], eventId[FK:Event.Eid]

Domain constraints:(rating >= 0 AND rating <= 5)

Default values: non

- Taxi_Reservation(Id, datetime, location, destination, status, creator_id[FK: METU_User:id])

We are expecting that each week 10 taxi reservations will be created thus yearly 500 taxi reservations.

Regarding the select queries, most of them will be related to the date and destination and location, status.

Update queries would be frequent on the status of taxi_reservation. The delete queries would be frequent as well depending on the status of the taxi reservation. However since the number of selects are more than the deletes and the update we have made a tradeoff and chosen to create an index for the status although the insert, update and delete slower and select faster.

There will be rare updates on the datetime but a lot of select queries for seeing new reservations thus we will create a secondary index for taxi reservation as well.

As for the destination and location the number of searches depending on these values will not be as many as the datetime ndand status, thus no need for an index for these attributes.

Summary:

Primary index for id

Secondary index : datetime, status

Not nulls: all attributes

Primary key: Id

Alternative key / unique: non

Foreign keys: creator_id[FK:METU_User:id]

Domain constraints: status IN ('Pending', 'Assigned', 'Canceled')

Default value: status: 'Pending'

-Reservation_Request(<u>taxi_id[FK:Taxi.vehicle_id], reservation_id[FK:Taxi_Reservation.id]</u>, price, status)

For each taxi reservation we are expecting that on average 2 requests to be created thus making the cardinality of this table 1000 annually.

Regarding the select queries, we expect that most of them will be related to the primary key. Update queries would be frequent on the status of reservation_Request. The delete queries would be frequent as well depending on the status of the reservation request.Thus only the primary index will be enough for the best performance.

Summary:

Primary index for taxi_id, reservation_id

Secondary index : non

Not nulls: all attributes

Primary key: taxi_id, reservation_id

Alternative key / unique: non

Foreign keys: taxi_id[FK:Taxi.vehicle_id], reservation_id[FK:Taxi_Reservation.id]

Domain constraints: status IN ('Pending', 'Accepted', 'Rejected', 'Canceled')

Default value: status: 'Pending'

Changes on the Database

While implementing the interface we have recognized that we need to include the car id that will be used in a request for an event in case the requester wants to drive the event. This we have added a new relation between the request and the car and also added a attribute to the request relation which help us to understand of the requester want to be a driver or simply a passenger. Thus as a result the request table has changed to:

- Request(<u>user_from</u>[FK:METU_User:id], <u>EID</u>[FK:Event.Eid], notes, time_stamp, req_status, requester_type, requester_car_id[FK:car.vaicle_id])

However other tables are untouched and this change does not affect the normalization or denormalization stages of our system and they remain the same. Also this change does not affect our choices on the indexes that were chosen previously.

As a result, the EER diagram is shown below in which the added changes are highlighted:

TODO:

The Interface

As mentioned before our system is a web-based application which can be used by several clients at the same time. Thus we have implemented a website with the help of languages such as PHP with MySQLi extension, JavaScript, HTML, CSS with Bootstrap framework. We have used PHP and MySQLi to connect to the database and implement the back end of the system. We utilized mostly Bootstrap framework to design our website and Javascript to make the website interactable.

When the website is first launched the page shown in Figure 2 is loaded for the users, after the user has selected the option of metu user or a taxi driver the corresponding sign up pages are shown to them (Figure 3 and Figure 4), if they don't have an account already they can sign or choose the option of log in (Figure 5 and Figure 6).

At the stages of login and signup related error and success messages are shown which can be seen in the Figure 7.

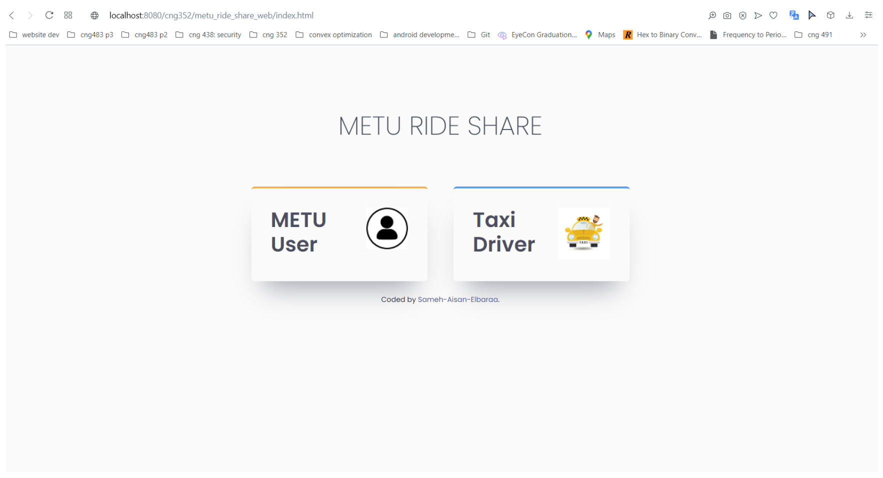
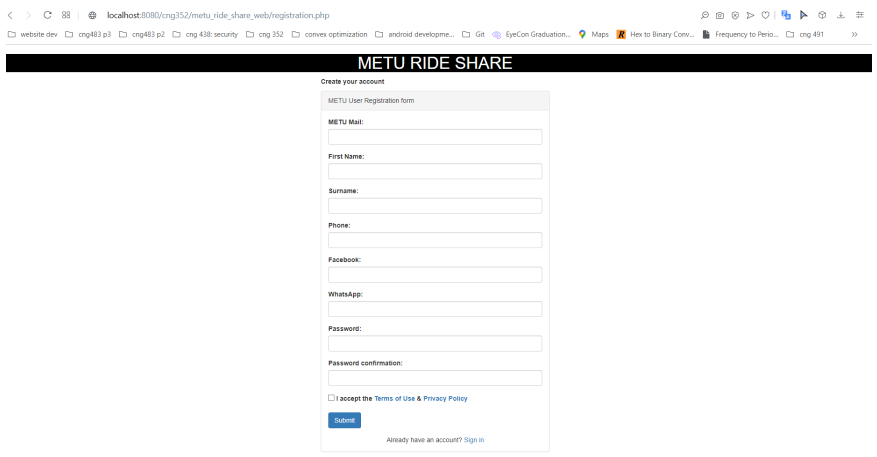

localhost:8080/cng352/metu_ride_share_web/index.html
METU RIDE SHARE
METU User
Taxi Driver
Coded by Sameh-Aisan-Elbaraa.

Figure 2


localhost:8080/cng352/metu_ride_share_web/registration.php

METU RIDE SHARE

Create your account

METU User Registration form

METU Mail:

First Name:

Surname:

Phone:

Facebook:

WhatsApp:

Password:

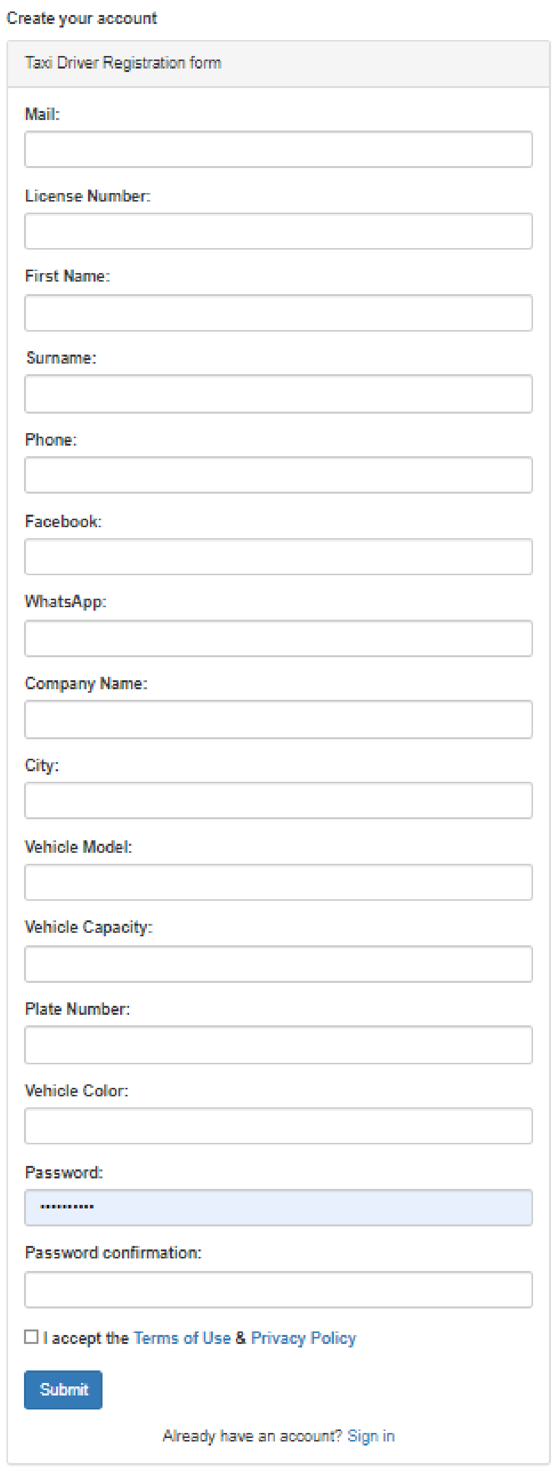
Password confirmation:

☐ I accept the Terms of Use & Privacy Policy

Submit

Already have an account? Sign in

Figure 3



Create your account

Taxi Driver Registration form

Mail:

License Number:

First Name:

Surname:

Phone:

Facebook:

WhatsApp:

Company Name:

City:

Vehicle Model:

Vehicle Capacity:

Plate Number:

Vehicle Color:

Password:

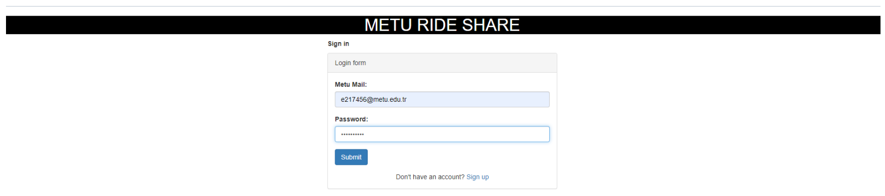
••••••••••

Password confirmation:

☐ I accept the Terms of Use & Privacy Policy

Already have an account? Sign in

Figure 4



METU RIDE SHARE

Sign in

Login form

Metu Mail:

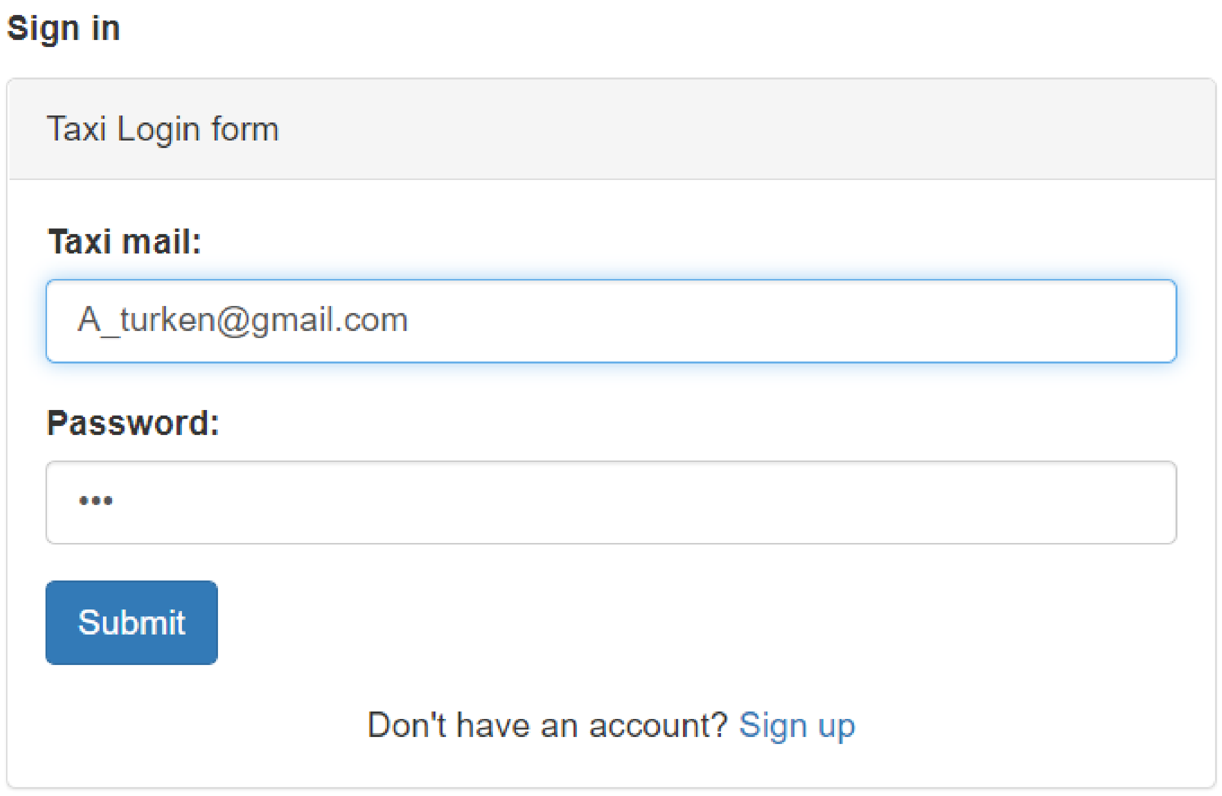
e217456@metu.edu.tr

Password:

••••••••••

Don't have an account? Sign up

Figure 5



Sign in

Taxi Login form

Taxi mail:

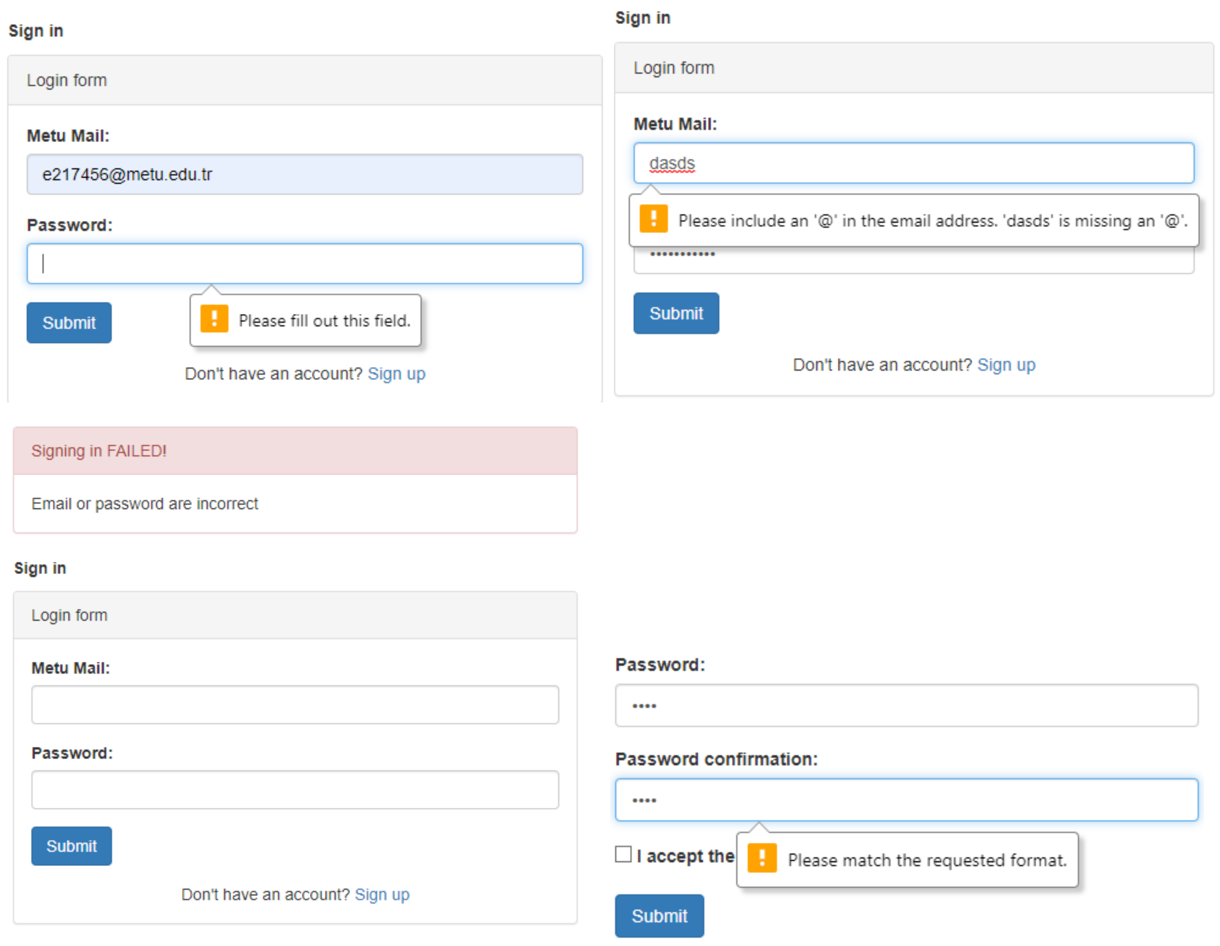
A_turken@gmail.com

Password:

•••

Don't have an account? Sign up

Figure 6


Sign in

Login form

Metu Mail:

e217456@metu.edu.tr

Password:

Submit

Please fill out this field.

Don't have an account? Sign up

Sign in

Login form

Metu Mail:

dasds

Please include an '@' in the email address. 'dasds' is missing an '@'.

Submit

Don't have an account? Sign up

Signing in FAILED!

Email or password are incorrect

Sign in

Login form

Metu Mail:

Password:

Submit

Don't have an account? Sign up

Password:

Password confirmation:

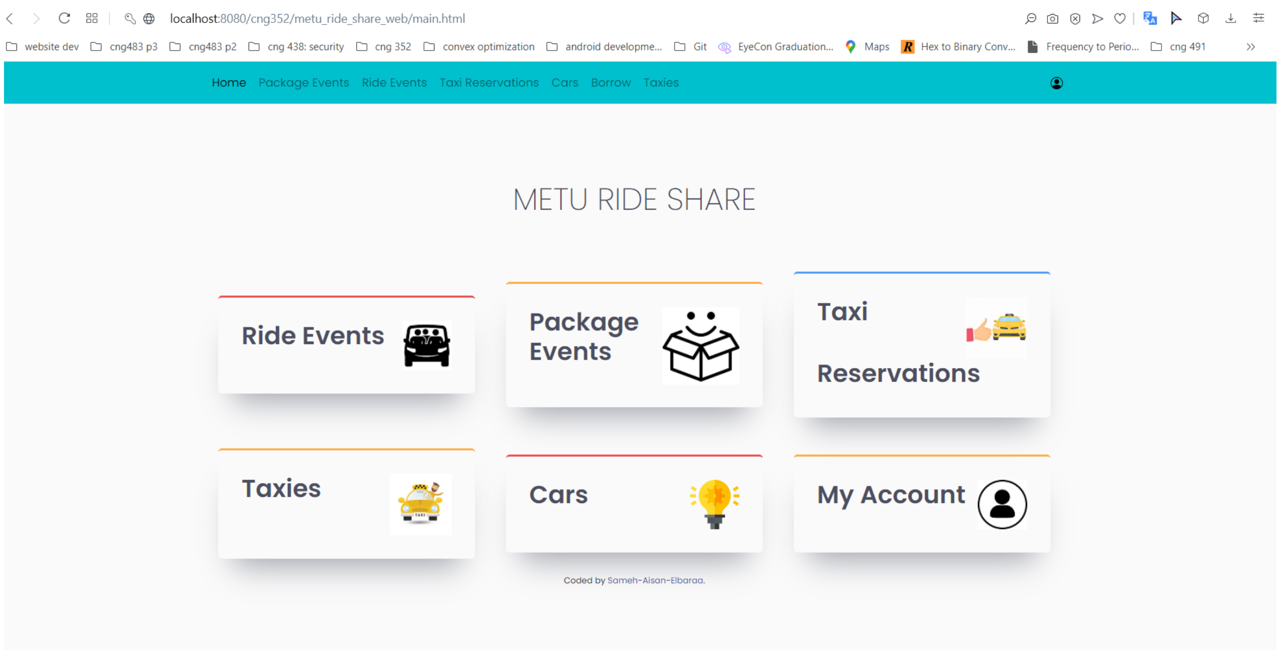
I accept the

Please match the requested format.

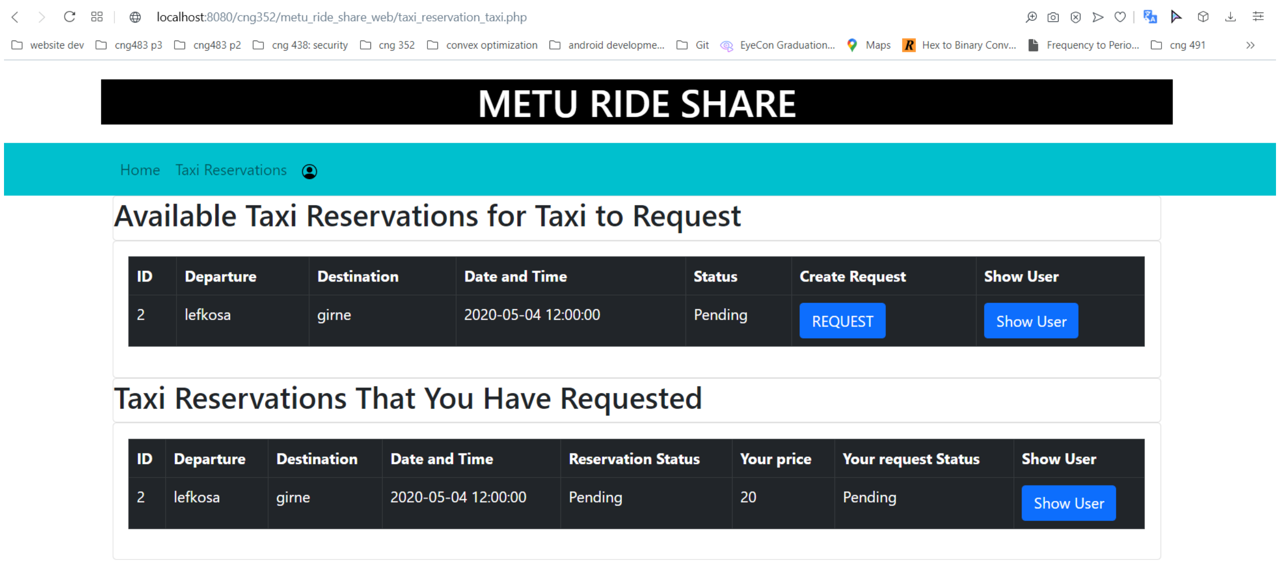
Submit

Figure 7

When the user logs in they are redirected to the main page of our website where they navigate to different pages by using the navigation bar on top (Figure 8).



When the taxi driver logs in they are redirected to the main page for the taxi driver. (Figure 9).


localhost:8080/cng352/metu_ride_share_web/taxi_reservation_taxi.php
METU RIDE SHARE
Home
Taxi Reservations

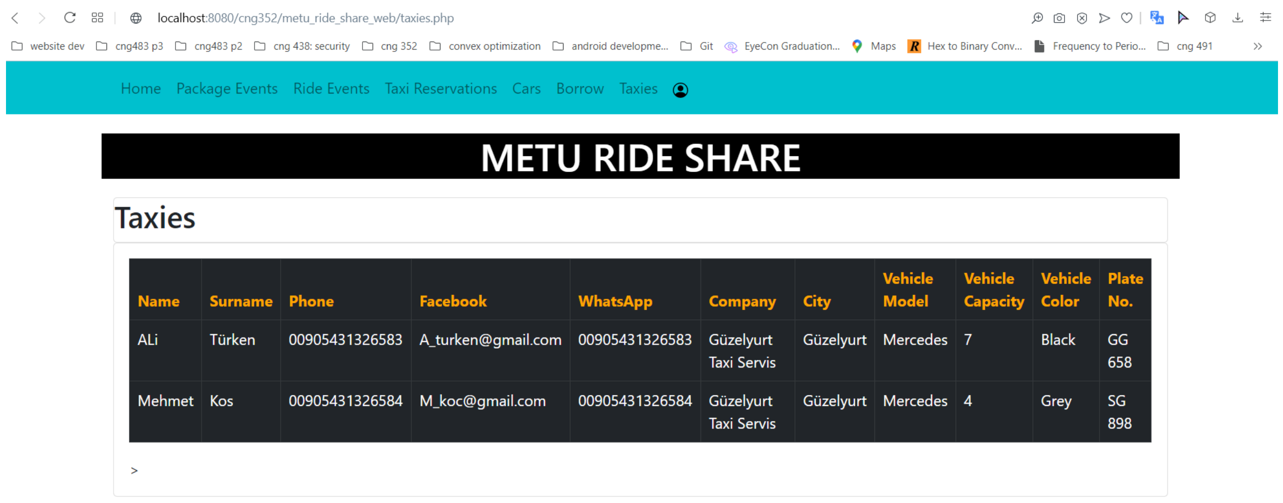
Available Taxi Reservations for Taxi to Request

ID	Departure	Destination	Date and Time	Status	Create Request	Show User
2	lefkosa	girne	2020-05-04 12:00:00	Pending	REQUEST	Show User

Taxi Reservations That You Have Requested

ID	Departure	Destination	Date and Time	Reservation Status	Your price	Your request Status	Show User
2	lefkosa	girne	2020-05-04 12:00:00	Pending	20	Pending	Show User

The Metu Users can see a list of taxies that are registered to the system.



localhost:8080/cng352/metu_ride_share_web/taxies.php

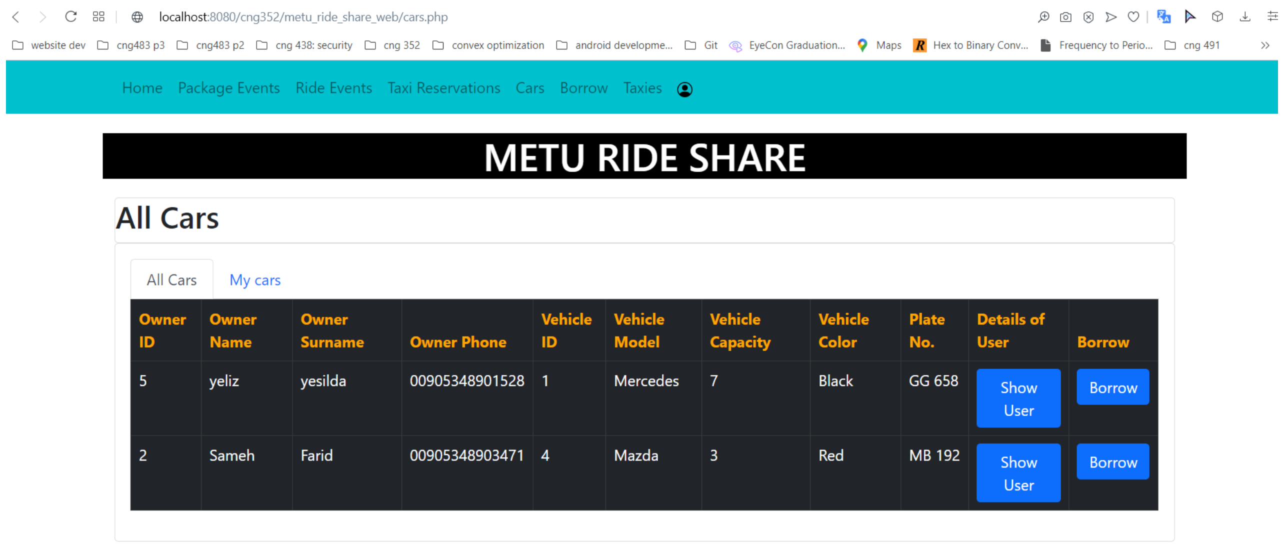
METU RIDE SHARE

Taxies

Name	Surname	Phone	Facebook	WhatsApp	Company	City	Vehicle Model	Vehicle Capacity	Vehicle Color	Plate No.
ALi	Türken	00905431326583	A_turken@gmail.com	00905431326583	Güzelyurt Taxi Servis	Güzelyurt	Mercedes	7	Black	GG 658
Mehmet	Kos	00905431326584	M_koc@gmail.com	00905431326584	Güzelyurt Taxi Servis	Güzelyurt	Mercedes	4	Grey	SG 898

>

They can see a list of cars which could help them to borrow:



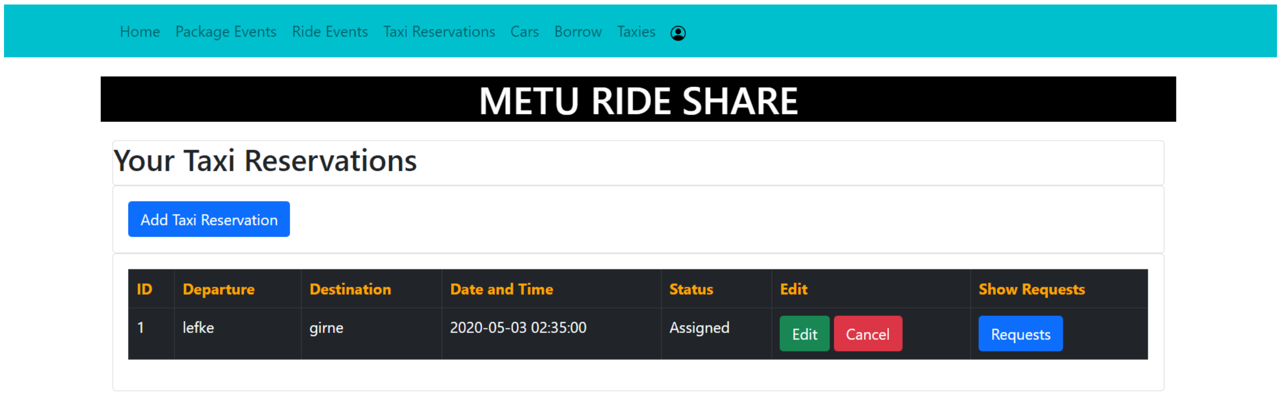
localhost:8080/cng352/metu_ride_share_web/cars.php

METU RIDE SHARE

All Cars

Owner ID	Owner Name	Owner Surname	Owner Phone	Vehicle ID	Vehicle Model	Vehicle Capacity	Vehicle Color	Plate No.	Details of User	Borrow
5	yeliz	yesilda	00905348901528	1	Mercedes	7	Black	GG 658	Show User	Borrow
2	Sameh	Farid	00905348903471	4	Mazda	3	Red	MB 192	Show User	Borrow

They can see a list of their taxi reservation, cancel, edit and add.


Home Package Events Ride Events Taxi Reservations Cars Borrow Taxies

METU RIDE SHARE

Your Taxi Reservations

Add Taxi Reservation

ID	Departure	Destination	Date and Time	Status	Edit	Show Requests
1	lefke	girne	2020-05-03 02:35:00	Assigned	Edit Cancel	Requests

SUMMARY OF THE INDEXES IN MYSQL

Use the same datatype for comparing

When you create a table with a primary key or unique key, MySQL automatically creates a special index named PRIMARY. This index is called the clustered index.

Once a clustered index is created, all rows in the table will be stored according to the key columns used to create the clustered index.

Because a clustered index store the rows in sorted order, each table have only one clustered index.

If you do not have a primary key for a table, MySQL will search for the first UNIQUE index where all the key columns are NOT NULL and use this UNIQUE index as the clustered index.

In case the InnoDB table has no primary key or suitable UNIQUE index, MySQL internally generates a hidden clustered index named GEN_CLUST_INDEX on a synthetic column which contains the row ID values.

As a result, each InnoDB table always has one and only one clustered index.

All indexes other than the clustered index are the non-clustered indexes or secondary indexes. In InnoDB tales, each record in the secondary index contains the primary key columns for the row as well as the columns specified in the non-clustered index. MySQL uses this primary key value for the row lookups in the clustered index.

Therefore, it is advantageous to have a short primary key otherwise the secondary indexes will use more space. Typically, the auto-increment integer column is used for the primary key column.

-> the primary key should be small since it will be stored beside the secondary index

-> the look at the where operations in the select for the secondary index

REFINEMENTS:

1. **We can add the status to the taxi reservation**
2. **We may add some attributes to the the many to many relations**
3. **We may combine the drive by and delivered by form package and the ride to the event**

TODO:

1. The taxi reservation should have a destination so that the price can be set by the taxi driver when they request
2. We have to decide on domains of the status related to : taxi reservation, reservation request, request, event, borrow (note the cancel is bu the person who created the requests or the events or the borrow requests)
    a. Taxi reservation request status: pending, accepted, rejected, canceled
    b. Taxi reservation status: pending, assigned, canceled
    c. Request status: pending, accepted, rejected, canceled
    d. Event status: active, full, canceled
    e. Borrow: pending, accepted, rejected, canceled
3. Check for the extra constraints that we can add
    a. The type of the constraints that we can have:
    b. Not null
    c. Unique
    d. PK
    e. FK
    f. Check
    g. default
4. We have to decide on the domain constraints
5. Related to index 1: change the metu_mail from the primary key and change all other selects...
6. We should add the delete operations: delete a request, delete a event or just do the cancel option in the status
7. We have to take care of the number of seats so that the system can check
8. The abbreviations of the status domain should be added
9. Discuss the workload of your application
10. the frequency of the queries of the update and select
11. The frequency of the delete and insert
12. the approximate number of records in your tables, the way it will progress
13. The package ID is added in the implementation, should be discussed and added to the report
14. Status is added to the taxi_reservation, should be discussed and added to the report
15. Check the report/db for any inconsistency
16. We have to check the inserted data
17. We have to take care of the dummy data for the selection
18. Delete and update on cascade....

INDEX and CONSTRAINTS:

1. The primary keys should be small but the metu mail is very long: we should have another id for the users: it will not maybe benefit the secondary indexes but it will be maybe more helpful when doing the join on the metu_mail
2. Licence_plate_no can be UNIQUE but if we do the unique there will be a secondary index assigned to it. But there is no need. Since it will be used rarely by administrator in case of a problem or a complaint
3. Regarding mail in taxi: it is an alternative key for the taxi and it can be used for login. Thus there will be an index assigned to it automatically. but should we make it limit to some char because the mail is long
4. Why is "`password` varchar(10) NOT NULL default 'NaN'," in the taxi
5. The datetime in events and taxi_reservation will be used to see if the event is still available and has not been passed thus and index will be great
6. Index also for status specially the events and the taxi_reservation would be good but for the request since when the taxi reservation and event is specified in the results of the join the number of request is very limited to on avg 5 so no need there.

QUESTIONS:

1. Do we need denormalization to not lose points: no as long as you explain
2. We have AVG and COUNT in select and update , do we need the SET for complexity: NO
3. We have simple insert operations should we do complex ones, also for the update: NO
4. Should we write queries that will be done by the administrator of the system
5. Should we sow calculations for why we chose a index as in the optimization lectures (then how to get the level of the Btree used for the indexes in MySQL): no
6. Should we talk about the transactions, transaction throughput, reponstime, disk storage (p37 tuning lecture) : no
7. Should we know about the file organization of the mysql for innoDB: no
8. How to store the password for the users
9. Related to todo 11: should we discuss the ones that we wrote and its relation to the tables: yes an estimation
10. Related to to 10: what is the work load: the number of rows an dthe frequencies of the queries then decide on the index
11. Since all the queries are not written yet how can we specify how many times the tables will be accessed: do an estimation and write the ones that will be used.

LOST PONT IN STEP 4:

1. **We did the one to one relation**
2. **We saved extra information about the rating and the no_reviews**
3. **We added extra status column for the events that are full, past**