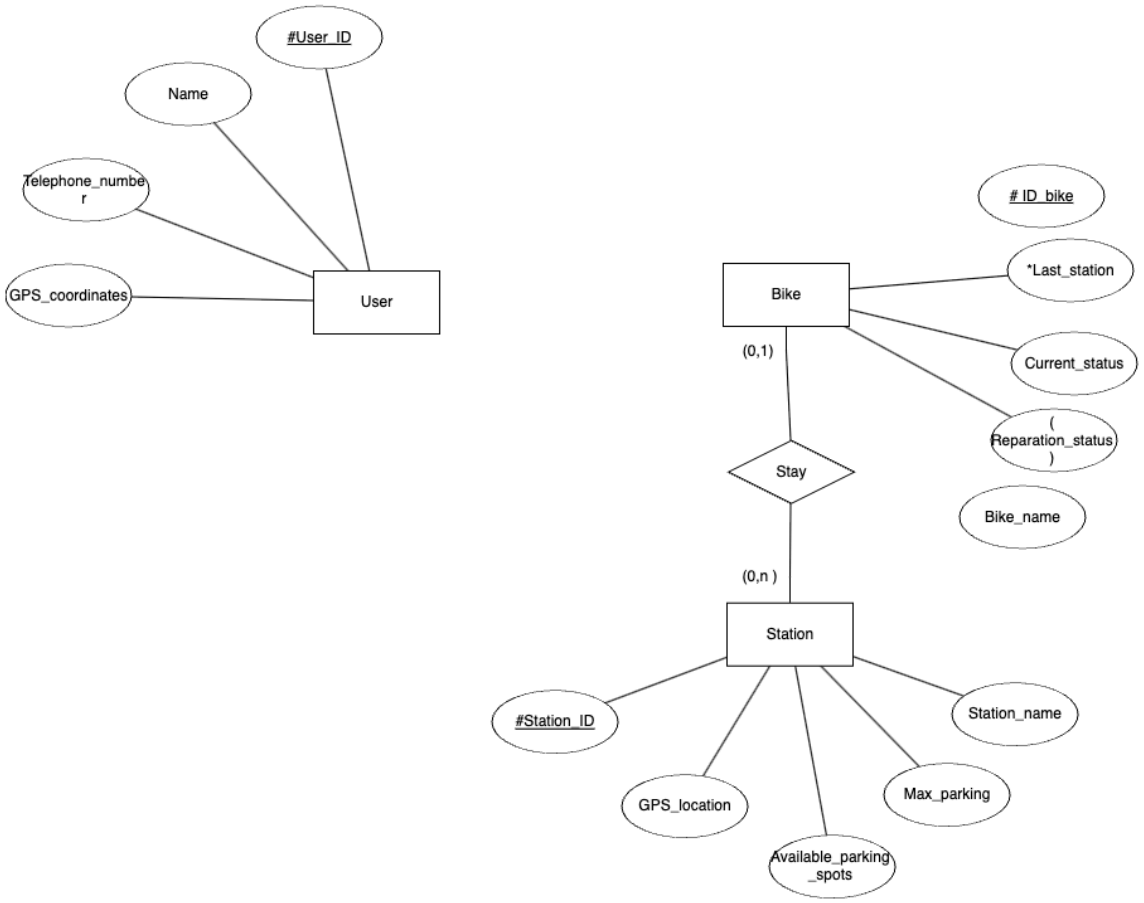
OBLIG 1.

Jeg har jobbet med @marknu0536@uib.no

Problem 1.

1.1

User har ingen relasjoner til de andre.
Bike og station har en



1.2

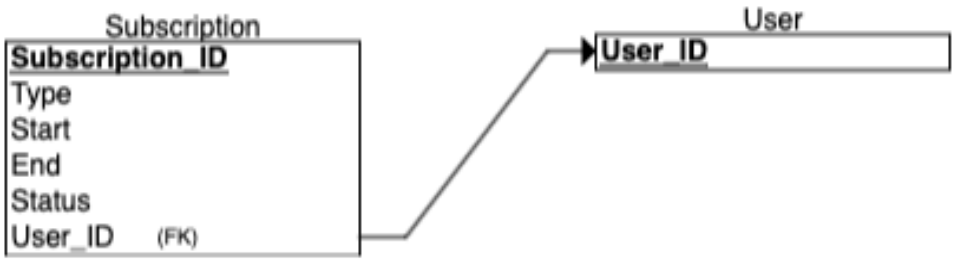
User ←→ Bike: Ingen relasjon eller eierskap
User ←→ Station: Ingen relasjon eller eierskap
Bike ←→ Station: Har ett forhold *stay*, i dette tilfelle har *station* eierskap over *bike,* siden *bike* har *attributen *Last_station* som er en FK, og *station* har *Station_ID* som er PK.

Problem 2

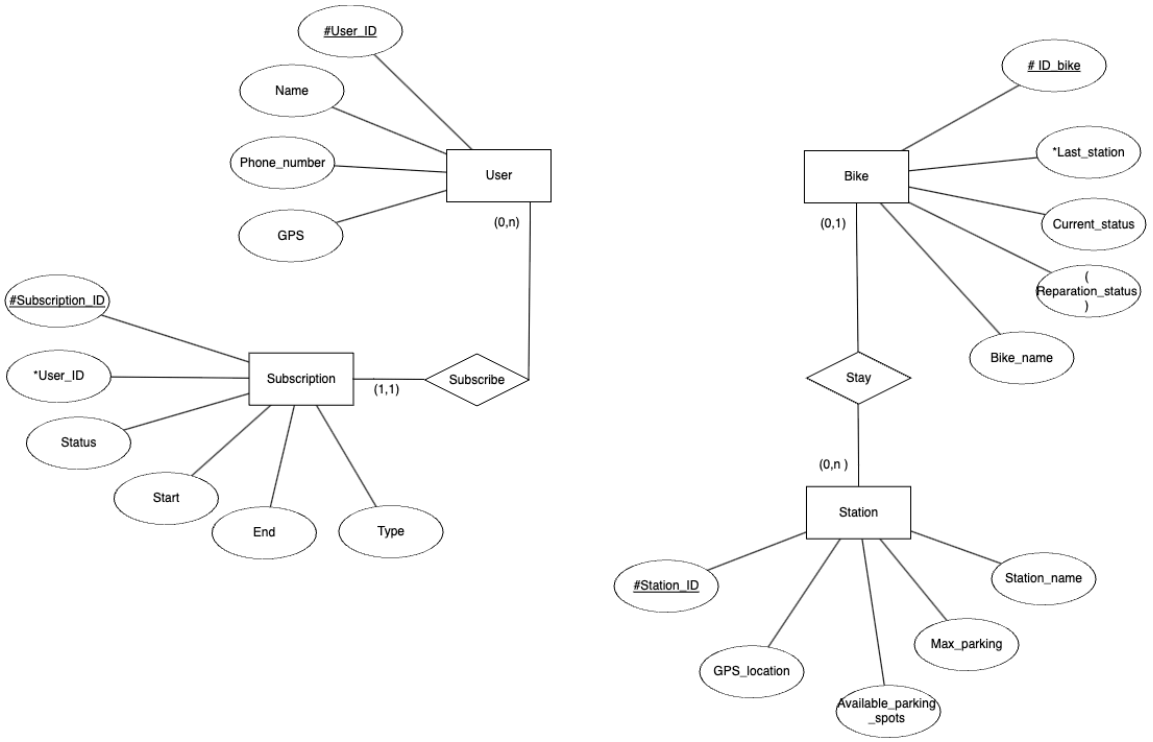
2.1

Hovedproblemet ved metoden foreslått er at vi ikke har mulighet til å ha flere subscriptions per user, en user kan kun ha en unik subscriptions id og flere vil ikke være mulig. Hvis en bruker avslutter subscription og starter en ny går hele systemet i kluss. I tillegg er det uoversiktlig å la status, start, end og type flyte rundt som attributes til user, når de bør være egenskapene til en subscription. Start, end og type kunne vært start på da brukeren lagde useren, dette kan også skape miskommunikasjon og forvirring.

2.2



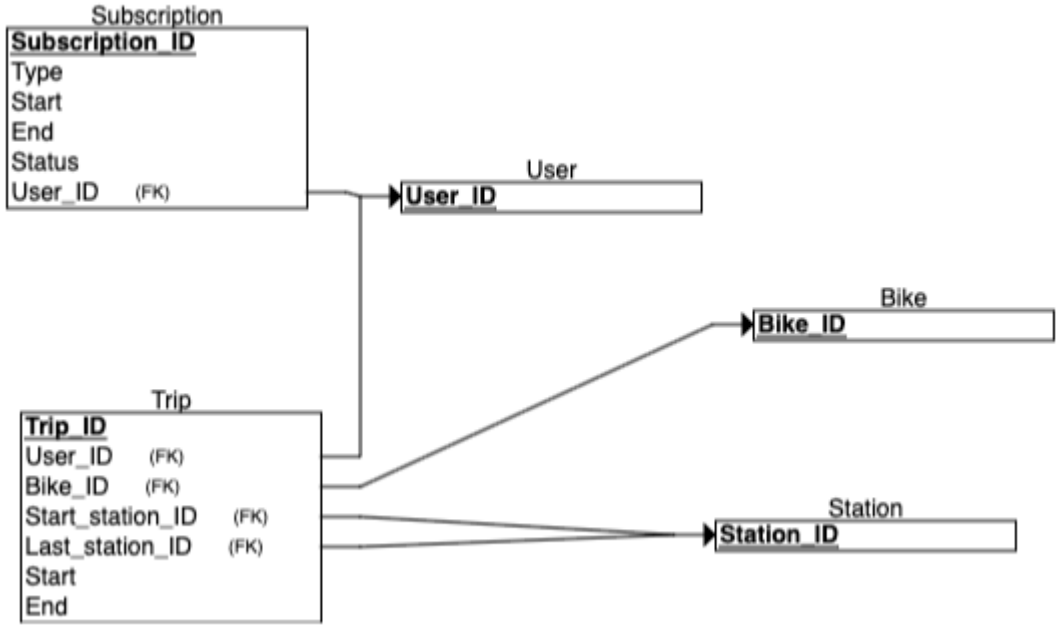
2.3



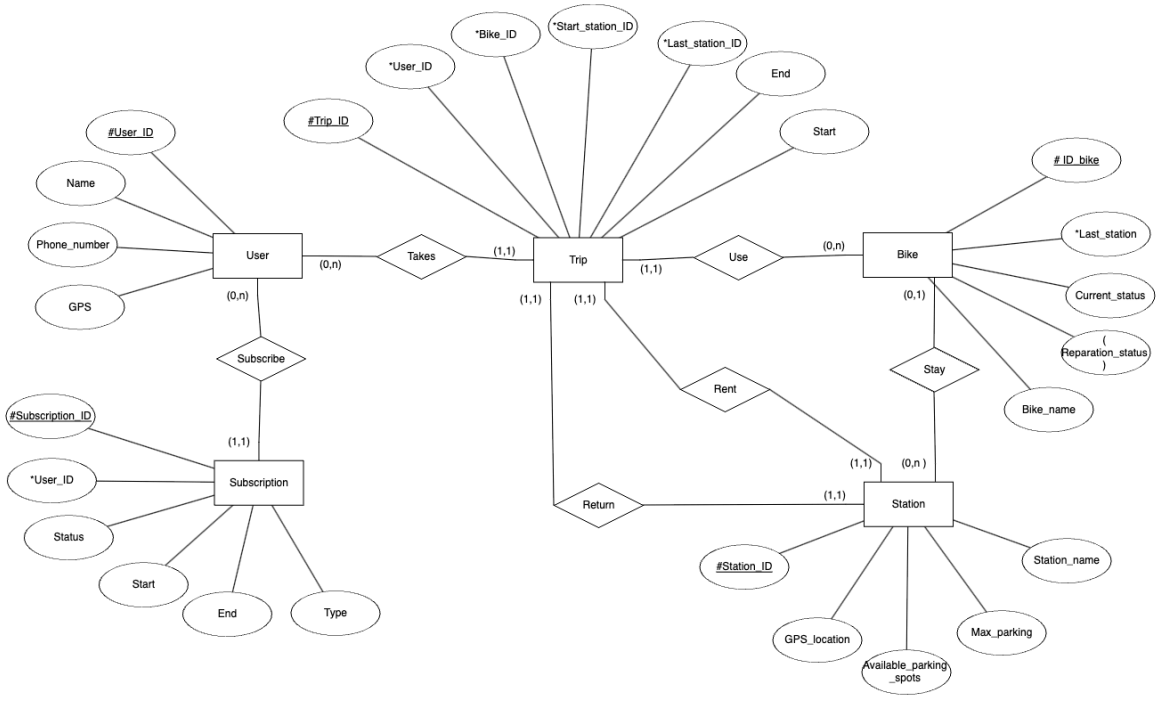
Problem 3

3.1

Jeg brukte ER-Plus sin tabell-model, syntes det var mer oversiktlig.



3.2



3.3

For å holde databaser oversiktlige, enklere å vedlikeholde og minimere duplisering, brukes databasenormalisering. Hver høyere form legger til flere krav for å oppnå det neste. Normalisering gjør det enklere å oppdatere databasen uten feil, i tillegg til å holde informasjonen konsistent.

1NF:

Hvis hver kolonne har kun verdi, altså at hver attribut i en relasjon kun har en verdi, er den i 1NF.
Vi kan skrive dette om til tre separate krav:
1. Hver kolonne har en verdi.
2. Alle kolonnene har unike navn.
3. Rekkefølgen på datene og kolonnene har ingen betydning for dataintegriteten.

2NF:

Siden hver høyere form legger til flere krav for å oppnå den neste, er en database i 2NF hvis den er i 1NF i tillegg til:

Alle ikke nøkkel attributer må være fullt avhengig av en kandidat nøkkel. Det vil si at hvis kandidaten nøkkelen består av to kolonner, skal du ikke kunne finne verdien til noen av attributene med kun en av kolonnene. Dette kalles å være *fully functionally dependent*.

Hvis du har relasjon som ikke oppnår 2NF, kan du dele relasjonen opp i flere tabeller

3NF:

En relasjon er i 3NF hvis den oppfyller 2NF, i tillegg:
Ikke ha en transitiv avhengighet, altså at primær nøkkelen må definere alle ikke nøkkel attributer, og ikke nøkkler kan ikke være avhengige av andre attributter.

A→B
B→C
A→C

Her er A→C transitiv avhengig fordi vi ikke direkte kan finne C fra A, vi må bruke B for å finne C.
For å unngå dette må man dele relasjonen opp i flere tabeller.

BCNF:

En relasjon er i BCNF hvis den oppfyller 3NF, i tillegg:
Hvis X→Y må være en kandidatnøkkel. Altså skal det kun være mulig å finne verdien til de andre attributene i tabellen med primærenøkkelen. Dette betyr at alle attributtene i tabellen må være direkte avhengige av primærenøkkelen, og ikke avhengige av andre attributter

For å oppnå BCNF må vi dele relasjonen opp i flere tabeller.

(https://www.geeksforgeeks.org/first-normal-form-1nf/), og forelesningsnotater.

3.4

I ER-diagrammet vi har laget tidligere inneholder bike attributet reparation_status, denne kolonnen har ikke en atomisk verdi, dette bryter med 1NF. For å oppnå BCNF kan vi dele lage en ny tabell Complaints, slik.
Complaint(#Complaint_ID, *user_ID, *bike_ID, reperation_type)

Vi kan da fjerne reparation_status fra bike.

I tillegg inneholder GPS kolonnen hos user og stasjon to verdier, både longitude og latitude. Vi splitter gps koordinater opp i to kolonner hos begge.

user(#user_id, name, phone_number, reperation_type, user_longitude, user_latitude)
station(#station_id, station_name, available_parking_spots, max_parking_spots, user_longitude, user_latitude)

Dette er det eneste som manglet for å oppnå BCNF.

Problem 4.

4.1

Først opprettet jeg databasen ved å bruke touch-kommandoen i terminalen. Deretter laget jeg en .sql-fil der jeg definerte tabellene og attributtene. Jeg brukte denne .sql-filen til å legge strukturen inn i databasen.

I ZIP-mappen har jeg også lagt til en extra-mappe. Denne inneholder blant annet .sql-filen og andre filer som ble brukt i oppgaven.

4.2

For å fylle databasen med data har jeg laget et skript som tar inn to filer:

- En CSV-fil med dataene
- En databasefil der dataene skal lagres

Skriptet leser dataene fra CSV-filen og legger dem inn i databasen.

Jeg har et GitHub-repository som inneholder:

- Skriptet for å fylle databasen
- README.md med instruksjoner om hvordan du bruker skriptet
- Source-koden, som også ligger i en egen mappe kalt extra i ZIP-filen

Link til repoet:

GitHub - SKOLE/INF115/OBLIG-1

4.3

Se vedlagt fil oblig1.py.