## 0.1 Data model

En kritisk del af dette system er data storage og data retrieval. Dette er blevet implementeret i form af to relationelle databaser; en distribueret og en lokal.
Til den distribuerede database og til administrationshjemmesiden er et domænenavn blevet købt hos `www.unoeuro.com`, ved navn `www.trackabus.dk`. Herunde er databasen oprettet som en MySQL database på serveren `http://mysql23.unoeuro.com`
Den lokale database eksisterer, fordi brugeren skal kunne gemme busruter lokalt på sin telefon. Dette er blevet implementeret i form af en SQLite database.
Diagrammer kan findes i fuld størrelse i bilag under Diagrammer/Database Diagrammer

### 0.1.1 Design af MySQL database

Den distribuerede database gemmer alt information vedrørende busserne og deres ruter. Opbygningen af databasen kan ses som tre komponenter der interagerer; Busser, busruter og stoppesteder.
Samtlige komponenter er defineret ved positions data i form af punkter. Disse punkter er længde- og breddegrader og kan ses som den fysiske position af den komponent, de relaterer til. Disse falder derfor i tre katagorier; Busposition, rutepunkter med stoppesteder og waypoints.

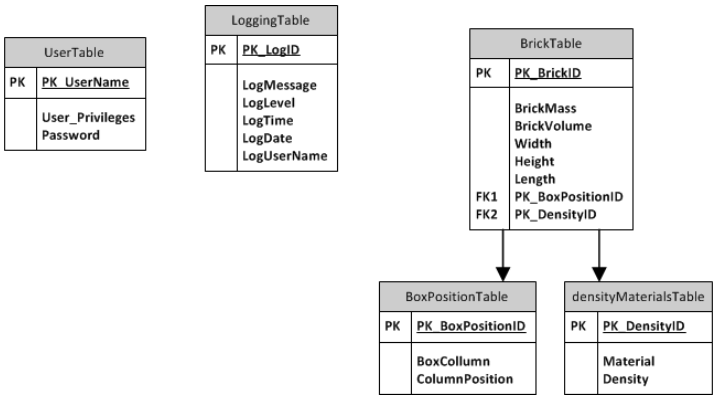
- Busposition er defineret som den fysiske placering af en given bus. I dette projekt var der dog ikke tilgang til nogen fysiske busser, så denne katagori af positions data blev simuleret. Simulatoren kunne dog skiftes ud med en virkelig bus, hvis position for denne kunne stilles til rådighed.

- Rutepunter og stoppesteder indeholder positionsdata, som bruges til at tegne ruten eller lave udregning på. Disse udregninger er defineret senere under "Stored procedures"og "Functions".

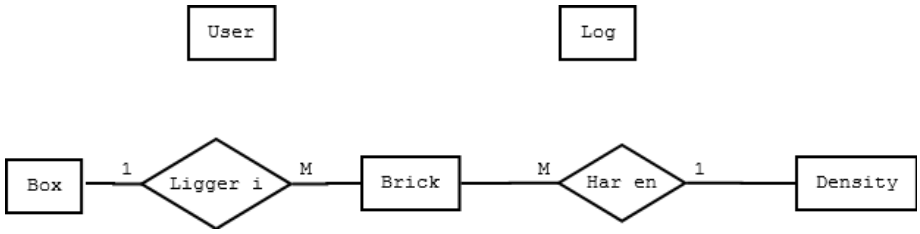
- Waypoints bruges som "genskabelses-punkter"til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet, til at genskabe den rute de beskriver.

Hele systemet er opbygget omkring oprettelse, fjernelse og manipulation af positions data. Dette er klart afspejlet i database i form hvor meget dette data bliver brugt.
Tidligt i udviklingsprocessen blev det fastsat at positions data have en præcision på seks decimaler, da dette ville resultere i en positions afvigelse på under en meter. Systemet virker stadig med en lavere præcision, men dette vil resultere i en større positionsafvigelse.

Databasen er bygget op af følgende tabeller: Bus, BusRoute, BusRoute_RoutePoint, BusRoute_BusStop, BusStop, GPSPosition, RoutePoint, Waypoint.
På figur 1 vises opbygningen af tabellerne som et UML OO diagram, og på figur 2 kan relationerne i databasen ses som et ER diagram.




Figur 1: UML OO diagram over den distribuerede MySQL database




Figur 2: ER Diagram over den distribuerede MySql database

Herunder følger en forklaring af tabellerne og deres rolle i systemet.

- **Bus**
  - Indeholder alt relevant data vedrørende kørende busser. fk_BusRoute er en foreign key til BusRoute tabellen og definerer hvilken rute bussen kører på.

IsDescending er et simpelt flag, som bestemmer i hvilken retning bussen kører. Hvis IsDescending er true, betyder det at bussen kører fra sidste til første punkt defineret ved ID i BusRoute_RoutePoint, og omvendt hvis den er false.
Som den eneste tabel er der mulighed for, at nulls kan fremkomme. Dette vil ske i situationer hvor bussen eksisterer i systemet, men endnu ikke er sat på en rute. Tabellens primary key er sat til at være det ID som defineres ved busses oprettelse. Dette nummer vil også stå på den fysisk bus.

- **BusRoute**
  - Indeholder detaljer omkring Busruten foruden dens rutepunkter. BusNumber er ikke nødvendigvis unikt, da en kompleks rute er bygget op af to eller flere underruter. Derfor bliver tabelens primary key sat til et autogeneret ID, som bliver inkrementeret ved nyt indlæg i BusRoute. BusNumber er rutenummeret, og også det nummer som vil kunne ses på bussens front. Nummeret er givet ved en varchar på 10 karakterer, da ruter også kan have bogstaver i deres nummer. Hvis SubRoute er sat til nul, vil ruten kun bestå af det enkelte ID, men hvis ruten er kompleks vil SubRoute starte fra et, og inkrementere med en for delrute på den givne rute. Ruter er i denne sammenhæng defineret som turen mellem to endestationer, og hvis en rute har mere end to endestation, vil den have minimum to hele ruter sat på det givne rutenummer.

- **BusRoute_RoutePoint**
  - Indeholder den egentlige rute for det givne rutenummer.Primary keyen er IDet i denne tabel og autogenereret, men bruges til at definere rækkefølgen på punkterne, som ruten bliver opbygget af. fk_BusRoute er foreign key til IDet for busruten, og fk_RoutePoint er foreign key til IDet for rutepunktet på et givet sted på ruten. Det første og sidste punkt for den givne rute vil altid være de to endestationer på ruten.
    Rutepunkterne for stoppestedet bliver lagt ind i listen ved hjælp af en forklaret i afsnittet "IMPLEMENTERING: ADMINISTRATOR SIDE".

- **BusRoute_BusStop**

  - Indeholder stoppestedsplanen for det givne rutenummer. IDet i denne tabel er autogeneret, men bruges til at definere rækkefølgen på stoppestederne på den givne rute. fk_BusRoute refererer til den busrute stoppestedet er på, og fk_BusStop refererer til selve stoppestedet. Det første og sidste ID for den givne busrute, vil være de to endestationer på den givne rute.

- **BusStop**

  - Indeholder alle stoppesteder i systemet. Primary keyen er IDet i denne tabel og er autogeneret. StopName er navenet på det givne stoppested, og er en varchar på 100 karakterer.
    fk_RoutePoint er en foreign key til IDet i RoutePoint tabellen, og vil være det fysiske punkt for stoppestedet givet ved en længde- og breddegrad.

- **RoutePoint**

  - Indeholder alle punkter for alle ruter og stoppesteder. Primary keyen er sat til at være et autogeneret ID. Hvert indlæg i denne tabel vil definere en position på verdenskortet. Longitude og latitude er i denne sammenhæng længde- og breddegraden, og de er defineret ved en number med 15 decimaler. Alle 15 decimaler er ikke nødvendig i brug og ved en indsættelse af et tal på f.eks. 6 decimaler, vil de sidste 9 være sat til 0.

- **GPSPosition**

  - Indeholder alle kørende bussers position. Primary keyen er sat til et ID, som bruges til at definere rækkefølgen på indlægene, således det højeste ID for en given bus vil være den nyeste position. Longitude og Latitude er Længde- og Breddegraden for den givne bus. Både Longitude og Latitude er givet ved 15 decimaler, dog hvor alle 15 ikke nødvendigvis er i brug. Ved en indsættelse af et tal på f.eks. 6 decimaler, vil de sidste 9 være sat til 0. UpdateTime er et timestamp for positionen og bruges til, at udregne hvor lang tid bussen har kørt. Dette er beskrevet nærmere i afsnittene "Stored procedures"og "Functions". fk_Bus er en foreign key til tabellen Bus og bruges til at definere hvilken bus der har lavet opdateringen.

- **Waypoint**
  - Indeholder alle punkter der er nødvendige for genskabelse af en rute på administrations siden. Primary keyen er IDet og autogeneret. Den Bruges ikke ti andet en at unikt markere punktet.
    Longitude og Latitude er Længde- og Breddegraden for det givne punkt. Både Longitude og Latitude er givet ved 15 decimaler, dog hvor alle 15 ikke nødvendigvis er i brug. Ved en indsættelse af et tal på f.eks. 6 decimaler, vil de sidste 9 være sat til 0. fk_BusRoute er en foreign key til BusRoute tabellen, og definerer således hvilken BusRute det givne waypoint er relateret til.

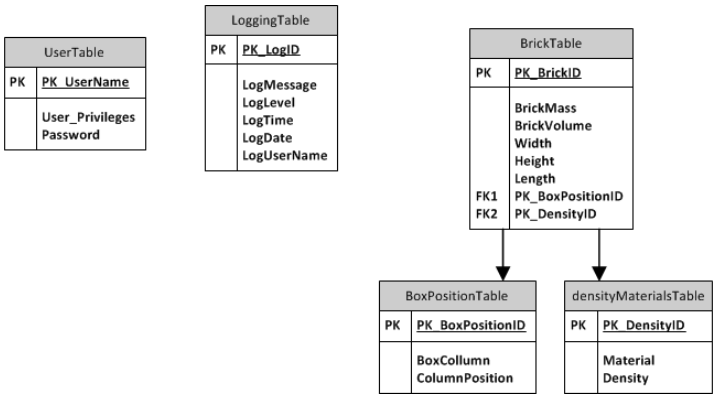
**Normalform**

Databasen er normaliseret til tredje normalform, hvor nulls er tilladt i enkelte tilfælde da det sås som gavnligt. Tabellen Bus indeholder alle oprettede busser, men det er ikke et krav, at en bus er på en rute. I tilfælde af en bus uden rute, vil fk_BusRoute og IsDescending være null.
Det antages at tredje normalform er tilstrækkeligt for systemet.
Begrundelsen for, at databasen er på tredjenormalform er:

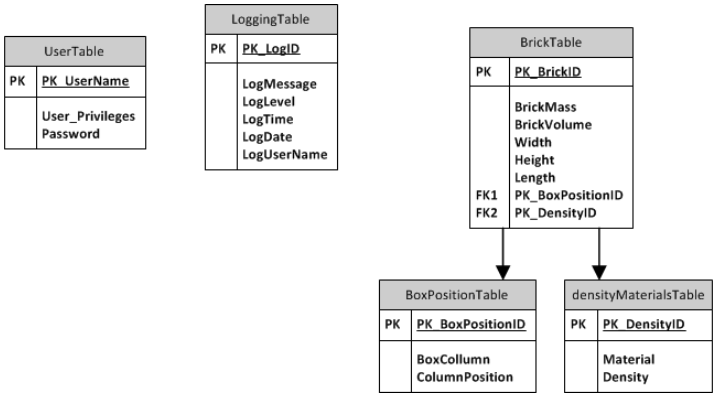
- Ingen elementer er i sig selv elementer. Dvs. ingen kolonner gentager sig selv.
- Ingen primary keys er composite keys, og derfor er ingen ikke keys afhængig af kun en del af nøglen
- Ingen elementer er afhængigt af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme tabel.

### 0.1.2 Design af SQLiteDatabase database

Mobil applikationen har en favoriserings funktion der bruges til at persistere brugervalgte ruter lokalt. Dette er gjort så brugeren hurtigt kan indlæse de ruter som bruges mest. Ruterne persisteres lokalt som et udsnit af den distribuerede database.
På figur 3 kan man se et UML OO diagram over den lokale SQLite database og på figur 4 kan man se et ER diagram over samme database.




Figur 3: UML OO diagram over den lokale SQLite database




Figur 4: UML OO diagram over den lokale SQLite database

Da den lokale database blot er et udsnit af den distribuerede MySQL database, henvises der til tabel beskrivelserne for MySQL tabellerne i forrige afsnit. Databasen er derfor også på tredje normalform, som MySQL databasen.

Den eneste forskel fra MySQL databasen er, at denne tabel gør brug af Delete Cascades. Dette vil sige, at sletningen af data fra SQLite databasen kun kræver at man sletter fra BusRoute og RoutePoint tabellerne, da disse har foreign keys i de andre tabeller. Da flere ruter med de samme stoppesteder godt kan indskrives er det blevet vedtaget, at stoppestederne ikke slettes, når en rute ufavoriseres. Dette betyder at stoppestederne kan genbruges ved nye favoriseringer.

### 0.1.3 Stored procedures

Der eksisterer kun Stored Procedures på MySQL database siden, og derfor vil dette afsnit kun omhandle disse.

Der er blevet lavet tre Stored Procedures i sammenhæng med tidsudregning for tætteste

bus til valgt stoppested. Disse tre vil blive beskrevet herunder, givet sammen med et kodeudsnit. I kodeudsnittet vil ingen kommentarer være tilstede. For fuld kode henvises der til bilags CDen, i filen Stored Procedures under Kode/Database.
I kodeudsnittene fremkommer forkortelserne "Asc"og "Desc. Dette står for Ascending og Descending og er en beskrivelse af, hvordan ruten indlæses. Ascending betyder at busruten indlæses fra første til sidste punkt i BusRoute_RoutePoint tabellen og Descending betyder at den indlæses fra sidste til første punkt.
Temporary tabeller bliver brugt meget i funktionerne og procedurene. De beskriver en fuldt funktionel tabel, med den forskel, at de kun er synlige fra den givne forbindelse. Når der i proceduren kun laves indskrivninger i temporary tables, gør det tilgangen trådsikker. Dette betyder at proceduren godt kan tilgås fra flere enheder på samme tid.

### CalcBusToStopTime

Denne Stored procedure er kernen i tidsudregningen. Den samler alle værdierne sender dem videre i de forskellige funktioner. På kodeudsnit 2 ses et udsnit af proceduren. I den fulde procedure, vil udregningerne for begge retninger hen til stoppestedet foregå, men da dette blot er en duplikering af samme kode, med forskellige variabler og funktionsnavne, vises dette ikke. Alle deklareringer af variabler er også fjernet.

**Kodeudsnit 1: CalcBusToStopTime. Finder nærmeste bus og udregner tiden begge veje**

```
1 create procedure CalcBusToStopTime(
2 IN stopName varchar(100), IN routeNumber varchar(10),
3 OUT TimeToStopSecAsc int, OUT TimeToStopSecDesc int,
4 OUT busIDAsc int, out busIDDesc int,
5 OUT EndBusStopAsc varchar(100), OUT EndBusStopDesc varchar(100))
6
7 BEGIN
8 drop temporary table if exists possibleRoutes;
9 create temporary table possibleRoutes(
10   possRouteID int,
11   possRouteStopID int
12 );
13
14 insert into possibleRoutes
```

```
15 select distinct BusRoute.ID, BusRoute_RoutePoint.ID from BusRoute
16 join BusRoute_BusStop on BusRoute.ID = BusRoute_BusStop.↩
     fk_BusRoute
17 join BusStop on BusRoute_BusStop.fk_BusStop = BusStop.ID
18 join BusRoute_RoutePoint on BusStop.fk_RoutePoint = ↩
     BusRoute_RoutePoint.fk_RoutePoint
19 where BusRoute.RouteNumber = routeNumber and BusStop.StopName = ↩
     stopName;
20
21 call GetClosestBusAscProc(@ClosestEndEPIdAsc, @ClosestBDIstAsc, ↩
     @ClosestBIDAsc );
22 select @ClosestsEndPointIDAsc, @ClosestBDIstAsc, @ClosestBIDAsc
23 into ClosestEndPointIdAsc,ClosestBusDistanceAsc,ClosestBusIdAsc;
24
25 select CalcBusAvgSpeedAsc(ClosestBusIdAsc) into ↩
     ClosestBusSpeedAsc;
26
27 set TimeToStopSecAsc = ClosestBusDistanceAsc/ClosestBusSpeedAsc;
28 set busIDAsc = ClosestBusIdAsc;
29
30 select BusStop.StopName from BusStop
31 inner join BusRoute_BusStop on BusRoute_BusStop.fk_BusStop = ↩
     BusStop.ID
32 inner join Bus on BusRoute_BusStop.fk_BusRoute = Bus.fk_BusRoute
33 where Bus.ID = ClosestBusIdAsc Order by BusRoute_BusStop.ID desc ↩
     limit 1 into EndBusStopAsc;
34
35 drop temporary table possibleRoutes;
36
37 END$$
```

Proceduren modtager navnet på det valgt stop, samt det valgte rutenummer. Ved fuldent forløb vil den returnere tiden for den nærmeste bus til det valgte stop, den nærmeste bus samt endestationen for den nærmeste bus. Alt returneres parvist i form af begge retninger.

Først findes mulige ruter fra givet stoppesteds navn og rutenummer og indlægges i en Dette er nødvendigt i tilfælde af komplekse ruter, hvor mere end en rute kan have samme stoppested og rutenummer. Herefter kaldes den anden stored procedure, som beskrives senere i dette afsnit. Denne procedure returnerer tætteste rutepunkt, IDet for den tætteste bus, samt afstanden fra den nærmeste bus til stoppestedet. Herefter udregnes bussens gennemsnitshastighed ved kaldet til CalcBusAvgSpeedAsc, som bruger det fundne bus ID. Denne funktion beskrives dybere senere under afsnittet "Functions".

Tiden fra bussen til stoppestedet findes ved at dividere distancen med gennemsnitshastigheden (Meter / Meter/Sekund = Sekund).
Til sidst findes endestation, og tiden, bus IDet og endestation returneres.

**GetClosestBusAscProc og GetClosestBusDescProc** Da proceduren for begge retninger er meget ens, vil der kun vises et kodeudsnit for GetClosestBusAscProc. Dette kan ses på kodeudnsit **??**.
Alle kommentarer og deklareringer er fjernet for at give et bedre overblik over funktionalitetet af proceduren. En detaljeret forklaring, samt foreskellene mellem GetClosestBusAscProc og GetClosestBusDescProc, følger efter kodeudsnittet

**Kodeudsnit 2: CalcBusToStopTime. Udregner nærmeste bus samt distance til stop og nærmeste rutepunkt**

```
1 create procedure GetClosestBusAscProc(OUT busClosestEndPointAsc ↩
    int, Out routeLengthAsc float, OUT closestBusId int)
2 begin
3
4 drop temporary table if exists BussesOnRouteAsc;
5 create temporary table BussesOnRouteAsc(
6   autoId int auto_increment primary key,
7   busId int,
8   stopID int
9 );
10
11 insert into BussesOnRouteAsc (busId, stopID) select distinct Bus.↩
    ID, possibleRoutes.possRouteStopID from Bus
12 inner join possibleRoutes on Bus.fk_BusRoute = possibleRoutes.↩
    possRouteID
13 where Bus.IsDescending=false;
14
15 select count(busId) from BussesOnRouteAsc into NumberOfBusses;
16
17 while BusCounter <= NumberOfBusses do
18   select busId,stopID from BussesOnRouteAsc where autoId = ↩
       BusCounter into currentBusId,currentStopId;
19
20   select GetClosestEndpointAsc(currentBusId)
21     into closestEndPoint;
```

```
22
23   if(closestEndPoint <= currentStopId) then
24     select GPSPosition.Latitude, GPSPosition.Longitude from ↩
          GPSPosition where GPSPosition.fk_Bus = currentBusId
25       order by GPSPosition.ID desc limit 1 into busPos_lat, ↩
            busPos_lon;
26
27       select CalcRouteLengthAsc(busPos_lon, busPos_lat, ↩
            closestEndPoint, currentStopId) into currentBusDist;
28   else
29     set currentBusDist = 10000000;
30   end if;
31   if (currentBusDist < leastBusDist) then
32     set leastBusDist = currentBusDist;
33     set closestbID = currentBusId;
34     set closestEP = closestEndPoint;
35   end if;
36   set BusCounter = BusCounter + 1;
37 end while;
38 set busClosestEndPointAsc = closestEP;
39 set routeLengthAsc = leastBusDist;
40 set closestBusId = closestbID;
41
42 drop temporary table BussesOnRouteAsc;
43 END $$
```