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 er server blevet lejet hos UnoEuro[1]. Herunder 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, og fuld implementering kan findes under Kode/Database.

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. Da der under systemets udvikling ikke var adgang til nogen fysiske busser, blev denne katagori af positions data simuleret. Simulatoren kan dog skiftes ud med en virkelig bus, hvis positioner for denne kan stilles til rådighed.

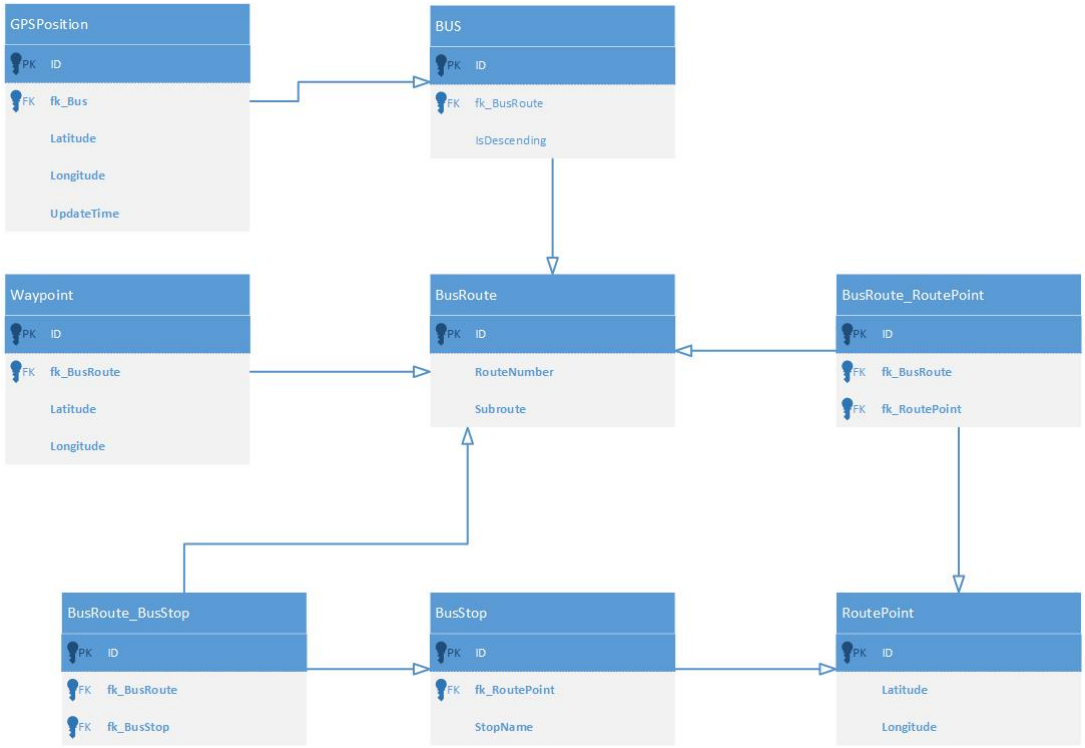
- Rutepunter og stoppesteder indeholder positionsdata, som bruges til at tegne- eller laver udregninger på ruten. Disse udregninger er defineret senere under afsnittene *9.1.3: Stored Procedures og 9.1.4: Functions*

- Waypoints bruges som genskabelses-punkter til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet

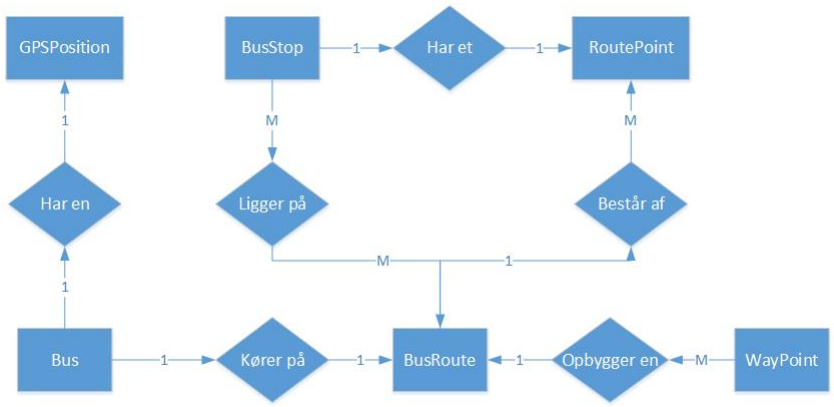
[1] www.unoeuro.com

Hele systemet er opbygget omkring oprettelse, fjernelse og manipulation af positions data. Dette er klart afspejlet i databasen i form af, hvor meget dette data bliver brugt.
Tidligt i udviklingsprocessen blev det fastsat at positions data skulle 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 og 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 fysiske bus.

- **BusRoute**
  - Indeholder detaljer omkring busruten foruden dens rutepunkter. BusNumber er ikke nødvendigvis en unik værdi, 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 ti 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 hver

delrute på den givne rute. Ruter er i denne sammenhæng defineret som vejen mellem to endestationer, og hvis en rute har mere end to endestationer, 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 ID'et i denne tabel og autogenereret, men bruges til at definere rækkefølgen på punkterne, som ruten bliver opbygget af. fk_BusRoute er foreign key til ID'et for busruten, og fk_RoutePoint er foreign key til ID'et 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 tilføjet til ruten ved hjælp af en funktion beskrevet i afsnittet *8.2.3: Komponent 3: Administrations hjemmeside.*
- **BusRoute_BusStop**
  - Indeholder stoppestedsplanen for det givne rutenummer. ID'et i denne tabel er autogeneret, men bruges til at definere rækkefølgen af 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 ID'et i denne tabel og er autogeneret. StopName er navnet på det givne stoppested, og er en varchar på 100 karakterer.
    fk_RoutePoint er en foreign key til ID'et 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 geografisk

position. Longitude og latitude er i denne sammenhæng længde- og breddegraden, og de er defineret som et decimal med 15 decimaler. Alle 15 decimaler er ikke nødvendigs i brug og ved en indsættelse af et tal på f.eks. seks decimaler, vil de sidste ni være sat til nul.

- **GPSPosition**
  - Indeholder alle kørende bussers nuværende og tidligere positioner. 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. seks decimaler, vil de sidste ni være sat til nul. UpdateTime er et timestamp for positionen og bruges til, at udregne hvor lang tid bussen har kørt. Dette er beskrevet nærmere i afsnittene *9.1.3: Stored Procedures og 9.1.4: 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 ID'et og autogeneret. Denne bruges ikke til andet end 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. seks decimaler, vil de sidste ni være sat til nul. fk_BusRoute er en foreign key til BusRoute tabellen, og definerer således, hvilken busrute det givne waypoint er relateret til.

Normalform

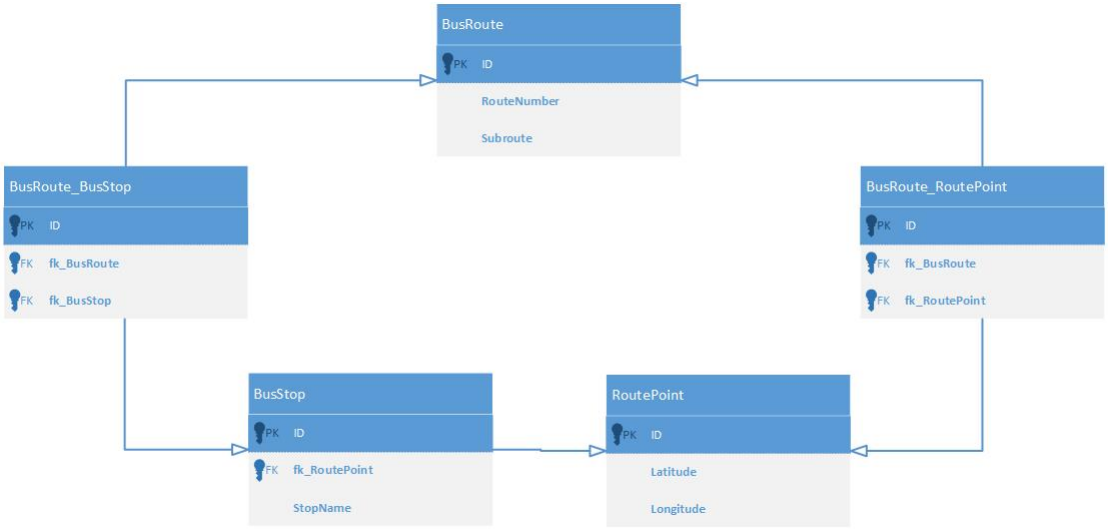
Databasen er normaliseret til tredje normalform, hvor nulls er tilladt i enkelte tilfælde da det sås som gavnligt. Tabellen Bus indeholder alle oprettede busser, men det er ikke et krav, at en bus er på en rute. I tilfælde af en bus uden rute, vil fk_BusRoute og IsDescending være null.

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 intet ikke-nøgle element afhængig af kun en del af nøglen
- Ingen elementer er afhængige af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme tabel.

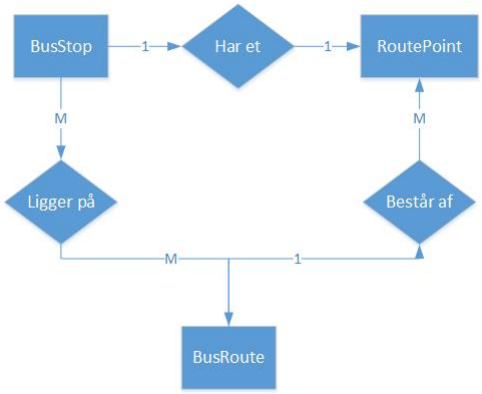
0.1.2 Design af SQLite databasen

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. Da den lokale database blot er et udsnit



Figur 3: UML OO diagram over den lokale SQLite database

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



Figur 4: ER diagram over den lokale SQLite database

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, udfra kodeudsnit. I kodeudsnittet vil ingen kommentarer være tilstede.
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 og BusRoute_BusStop tabellerne 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 tabeller, gør det tilgangen trådsikker. Dette betyder at proceduren godt kan tilgås fra flere enheder på samme tid. Når forbindelsen lukkes, vil de oprettede temporary tabeller slettes.

CalcBusToStopTime

Denne Stored Procedure er kernen i tidsudregningen. Den samler alle værdierne og sender dem videre i de forskellige funktioner. På kodeudsnit 1 ses et udsnit af proceduren. I den fulde procedure, vil udregningerne for begge retninger mod 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 inner join BusRoute_BusStop on BusRoute.ID = BusRoute_BusStop.↩
     fk_BusRoute
17 inner join BusStop on BusRoute_BusStop.fk_BusStop = BusStop.ID
18 inner join BusRoute_RoutePoint on BusRoute.ID = ↩
     BusRoute_RoutePoint.fk_BusRoute
19    and BusStop.fk_RoutePoint = BusRoute_RoutePoint.fk_RoutePoint
20 where BusRoute.RouteNumber = routeNumber and BusStop.StopName = ↩
     stopName ;
21
22 call GetClosestBusAscProc(@ClosestEndEPIdAsc, @ClosestBDIstAsc, ↩
     @ClosestBIDAsc );
23 select @ClosestsEndPointIDAsc, @ClosestBDIstAsc, @ClosestBIDAsc
24 into ClosestEndPointIdAsc,ClosestBusDistanceAsc,ClosestBusIdAsc;
25
26 select CalcBusAvgSpeedAsc(ClosestBusIdAsc) into ↩
     ClosestBusSpeedAsc;
27
28 set TimeToStopSecAsc = ClosestBusDistanceAsc/ClosestBusSpeedAsc;
```

```
29 set busIDAsc = ClosestBusIdAsc;
30
31 select BusStop.StopName from BusStop
32 inner join BusRoute_BusStop on BusRoute_BusStop.fk_BusStop = ↩
     BusStop.ID
33 inner join Bus on BusRoute_BusStop.fk_BusRoute = Bus.fk_BusRoute
34 where Bus.ID = ClosestBusIdAsc Order by BusRoute_BusStop.ID desc ↩
     limit 1 into EndBusStopAsc;
35
36 drop temporary table possibleRoutes;
37
38 END$$
```

Proceduren modtager navnet på det valgte stoppested, 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 mod stoppestedet.

Først findes samtlige mulige ruter udfra det medsendte stoppestedsnavn og rutenummer. Disse ruter indsættes i en temporary tabel. Dette er nødvendigt i tilfælde af komplekse ruter, hvor mere end en rute kan have samme stoppested og rutenummer. "GetClosestBusAscProc", som beskrives senere i dette afsnit. Denne procedure returnerer tætteste rutepunkt, ID'et 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 under afsnittet "Functions".

Tiden fra bussen til stoppestedet findes ved at dividere distancen med gennemsnitshastigheden (Meter / Meter/Sekund = Sekund).

Til sidst findes endestationen, og returneres sammen med tiden og bus ID'et.

GetClosestBusAscProc og GetClosestBusDescProc

Da proceduren for begge retninger er meget ens, vil der kun vises et kodeudsnit for "GetClosestBusAscProc". Denne kan ses på kodeudsnit 2.

Alle deklareringer er fjernet for at give et bedre overblik over funktionalitetet af proceduren. En detaljeret forklaring, samt forskellene mellem "GetClosestBusAscProc"og "GetClosestBusDescProc", følger efter kodeudsnittet.

Kodeudsnit 2: GetClosestBusAscProc. 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 $$
```

Denne procedure modtager ingen parametre, da den kun bruger data sat i "possibleRoutes"tabellen fundet i "CalcBusToStopTime"proceduren. Hovedfunktionaliteten i denne procedure er, at at udregne hvilken bus, der er tættest på det valgte stoppested. Dette repræsenteres ved bussens ID. Igennem denne udregning findes der også to underresultater der skal bruges i senere udregninger; Distancen fra bussen hen til stoppestedet, samt det tætteste rutepunkt bussen endnu ikke har nået.

Alle buser, som kører på en af de ruter i "possibleRoutes"og hvor "IsDescending"er sat til false udtages. Disse busser bliver parret med det ID stoppestedet har, i "BusRoute_RoutePoint"tabellen og et auto-inkrementeret ID startende fra 1. Disse værdier bliver lagt ind i "BussesOnRouteAsc"temporary tabellen. Herefter findes det antal af busser, der er blevet udtaget, og dette tal bruges til den øvre grænse for while-løkkens. Den nedre grænse er blot en counter som sættes til 1 ved initiering.

While-løkkens rolle er at iterere igennem samtlige busser, og udregne distancen fra hver bus til dens parrede stoppested, hvorefter der skal vælges den bus, der har den korteste distance til sit stoppested.

Først udregnes Det nærmeste rutepunkt ved et kald til funktionen "GetClosestEndpointAsc". Hvis dette rutepunkt har et større ID end busstoppets, vil distancen fra bussen til stoppestedet sættes til 10000000, altså meget højt. I en fysisk forstand vil dette ske, hvis bussen er kørt forbi det givne stoppested, og derfor ikke længere kan være den nærmeste bus til stoppestedet. Hvis rutepunktet derimod har et mindre ID end stoppestedet, vil de nyeste koordinater for bussen findes, og distancen fra bussen hen til stoppestedet vil udregnes ved et kald til funktionen "CalcRouteLengthAsc".

Herefter undersøges der, om den givne bus har en mindre distance hen til stoppestedet end den bus med den nuværende korteste distance. "leastBusDist"er sat til 100000, altså højt, men ikke lige så højt som det tal den nuværende distance sættes til, hvis bussen er

kørt forbi stoppestedet. Dette vil betyde at ingen sådan bus, ved en fejl, kan vælges som den tætteste bus. Hvis denne bus derimod har en mindre distance end den nuværende korteste distance, vil den mindste distance sættes til denne. ID'et for bussen samt det tætteste rutepunkt, vil også sættes i denne situation. Til sidst vil den korteste distance, det tætteste rutepunkt samt ID'et for den tætteste bus blive returneret.
I "GetClosestBusDescProc"(samme udregning, blot for rute der køre fra sidste til første stoppested), er der to definerende forskelle.
På kodeudsnit 3, kan den første forskel ses. I dette tilfælde hentes der kun busser ud hvor *IsDescending* er true, altså hvor den givne bus kører fra første til sidste stoppested.

Kodeudsnit 3: GetClosestBusDescProc forskel 1

```
1 ...
2 insert into BussesOnRouteDesc (busId,stopId) select distinct Bus.↩
    ID,      possibleRoutes.possRouteStopID from Bus
3 inner join possibleRoutes on Bus.fk_BusRoute = possibleRoutes.↩
    possRouteID
4 where Bus.IsDescending=true;
5 ...
```

På kodeudsnit 4, kan den anden forskel ses. Hvis en bus kører fra første til sidste stoppested, vil det nærmeste rutepunkt til bussen, have et større ID end stoppestedets, hvis bussen endnu ikke er kørt forbi. Derfor undersøges der her om rutepunktets ID er større eller ligmed stoppestedets ID, hvor der i "GetClosestBusAscProc"undersøges om det er mindre eller ligmed.

Kodeudsnit 4: GetClosestBusDescProc forskel 2

```
1 ...
2 if(closestEndPoint >= currentStopId) then
3 ...
```

0.1.4 Functions:

Igennem forløbet af "CalcBusToStopTime"proceduren, tages en del funktioner i brug. Disse bruges når kun en enkelt værdi behøves returneres. Funktionerne er delt om i to typer; Funktioner til udregning af relevant information til procedurene, samt matematik-funktioner. Der vil ikke vises kodeeksempler for matematik funktionerne i dette afsnit, men der henvises til afsnit *8.2.5: Komponent 5: Anvendt Matematik*, for beskrivelser af disse.

Som i afsnittet *9.1.3 Stored Procedures*, er funktionerne bygget op parvist; En funktion til busser der kører fra første til sidste stop (ascending), samt en anden til busser, der kører fra sidste til første stop (descending). Der vil kun vises et kodeudsnit af ascending-funktionerne, hvorefter forskellene i descending-funktionerne beskrives. Kodeudsnittene vil ikke indeholde kommentarer eller initialiseringer af variable, så et bedre overblik af funktionalitet kan gives.

GetClosestEndpointAsc og GetClosestEndpointDesc

Disse funktioner tages i brug i "GetClosestBusAscProc"og "GetClosestBusDescProc"procedurene, og bruges til at finde ID'et for det rutepunkt, en given bus er tættest på. Dette ID er dog ikke rutepunktets egentlige ID i "RoutePoint"tabellen, men derimod dens ID i "BusRoute_RoutePoint"tabellen. Denne bus er defineret ved dens ID, givet til funktionen som dens eneste parameter. På kodeudsnit 5 kan "GetClosestEndpointAsc"funktionen ses.

Kodeudsnit 5: GetClosestEndpointAsc finder det tætteste punkt på ruten fra bussen

```
1 create function GetClosestEndpointAsc(busID int)
2 returns int
3 begin
4 drop temporary table if exists ChosenRouteAsc;
5 create TEMPORARY table if not exists ChosenRouteAsc(
6   id int primary key,
7   bus_lat decimal(20,15),
8   bus_lon decimal(20,15)
9 );
10
11 insert into ChosenRouteAsc (id,bus_lat,bus_lon)
```

```
12 select BusRoute_RoutePoint.ID, RoutePoint.Latitude,  RoutePoint.↩
     Longitude from RoutePoint
13 inner join BusRoute_RoutePoint on BusRoute_RoutePoint.↩
     fk_RoutePoint = RoutePoint.ID
14 inner join Bus on Bus.fk_BusRoute = BusRoute_RoutePoint.↩
     fk_BusRoute
15 where Bus.ID = busID
16 order by(BusRoute_RoutePoint.ID) asc;
17
18 select ChosenRouteAsc.ID from ChosenRouteAsc order by id asc ↩
     limit 1 into RouteCounter;
19 select ChosenRouteAsc.ID from ChosenRouteAsc order by id desc ↩
     limit 1 into LastChosenID;
20
21 select GPSPosition.Latitude,  GPSPosition.Longitude from ↩
     GPSPosition where GPSPosition.fk_Bus = busID
22 order by GPSPosition.ID desc limit 1 into BusLastPosLat, ↩
     BusLastPosLon;
23
24 while RouteCounter < LastChosenID do
25   select bus_lon from ChosenRouteAsc where id = RouteCounter into↩
       R1x;
26   select bus_lat from ChosenRouteAsc where id = RouteCounter into↩
       R1y;
27   select bus_lon from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2x;
28   select bus_lat from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2y;
29   set BusDist = CalcRouteLineDist(BusLastPosLon, BusLastPosLat, ↩
      R1x, R1y, R2x, R2y);
30
31   if BusDist < PrevBusDist then
32     set PrevBusDist = BusDist;
33     set ClosestEndPointId = RouteCounter+1;
34   end if;
35   Set RouteCounter = RouteCounter + 1;
36 end while;
37 return ClosestEndPointId;
38 END$$
```

Ruten som den givne bus kører på hentes ud og gemmes i en temporary tabellen "ChosenRouteAsc". I denne tabel gemmes længde- og breddegrader, sammen det ID punktet har, i *BusRoute_ RoutePoint*-tabellen. Da samtlige punkter ligges ind i databasen samtidig, efter en rute er skabt på hjemmesiden, garanteres det, at punterne ligger sekvensielt. Punkterne gemmes altså i rækkefølge i *BusRoute_ RoutePoint*, uden spring i IDerne. Det-

te gør at punkterne kan itereres igennem, uden at der skal tages højde for spring, og kan sorteres efter ID i den rækkefølge der skal bruges (ascending for første til sidste stoppested, descending for sidste til første stoppested). Det er meget sandsynligt at det første ID hentet ikke er 1, Så det første og sidste punkt på ruten findes også, og bruges som den nedre og øvre grænse for while-løkken. På den måde vil der itereres igennem samtlige punkter på ruten, hvor ID'et for første og sidste stop ikke har nogen betydning for funktionen. Inden while-løkken startes, hentes bussens sidste position ud, så det ikke har nogen betydning, hvis bussens position ændrer sig under itereringen af ruten.
Så længe "routeCounter"(det nuværende ID der undersøges) er mindre end "LastChosenID"(Det sidste ID på ruten), udtages punkterne for det nuværende ID og det næste. Således laves der et linjestykke spændt ud mellem to punkter. Afstanden fra bussen til dens tætteste punkt på dette linjestyke, udregnes i "CalcRouteLineDist". Denne funktion er udelukkende matematisk og vil beskrives i *8.2.5: Komponent 5: Anvendt Matematik*. Hvis bussens position på et givent linjestykke ikke er gyldigt, vil 1000000, et stort tal, returneres. Dette tal vil være større end prevBusDist som har en initial værdi sat til 100000. Dette sørger for, at det givne endpoint ikke, ved en fejl, tælles med. Hvis den udregnede værdi af distancen til punktet på linjen, er mindre end den forrige distance, vil det næste punkt på ruten, i forhold til det punkt man undersøger, søttes til bussens tætteste. Ved en fuldent gennemiterering af ruten, vil bussens tætteste rutepunkt være fundet, og ID'et for dette punkt returneres.
I "GetClosestEndpointDesc"er der nogle enkelte forskelle, som her vil beskrives. På kodeudsnit 6, kan det ses hvordan ruten nu hentes ud i en tabel, hvor der sorteres efter ID'et i "BusRoute_RoutePoint"tabellen, i faldende rækkefølge.

Kodeudsnit 6: GetClosestEndpointDesc forskel 1

```
1 ...
2   insert into ChosenRouteDesc  (id,bus_lat,bus_lon)
3   select BusRoute_RoutePoint.ID, RoutePoint.Latitude,  RoutePoint↩
       .Longitude from RoutePoint
4   inner join BusRoute_RoutePoint on BusRoute_RoutePoint.↩
       fk_RoutePoint = RoutePoint.ID
5   inner join Bus on Bus.fk_BusRoute = BusRoute_RoutePoint.↩
       fk_BusRoute
```

```
6   where Bus.ID = busID
7   order by(BusRoute_RoutePoint.ID) desc;
8 ...
```

På kodeudsnit 7 ses det hvordan, der nu læses i modsat rækkefølge fra "ChosenRouteDesc"tabellen. "RouteCounter"er nu den øverste grænse, og "LastChosenID"er nu den nedre. Der læses nu også i omvendt rækkefølge fra tabellen, da ID'erne nu er faldende. Det vil også sige, at "RouteCounter"dekrementeres i stedet for inkrementeres i slutningen af hver iteration.

Kodeudsnit 7: GetClosestEndpointDesc forskel 2

```
1 ...
2 select ChosenRouteDesc.ID from ChosenRouteDesc order by id asc ↩
     limit 1 into LastChosenID;
3 select ChosenRouteDesc.ID from ChosenRouteDesc order by id desc ↩
     limit 1 into RouteCounter;
4 ...
5 while RouteCounter > LastChosenID do
6   select bus_lon from ChosenRouteDesc where id = RouteCounter ↩
       into R1x;
7   select bus_lat from ChosenRouteDesc where id = RouteCounter ↩
       into R1y;
8   select bus_lon from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2x;
9   select bus_lat from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2y;
10 ...
11 Set RouteCounter = RouteCounter - 1;
12 ...
```

CalcRouteLengthAsc og CalcRouteLengthDesc

Disse funktioner tages i brug i "GetClosestBusAscProc og "GetClosestBusDescProc"procedurene. De bruges til at udregne afstanden fra en bus til det valgte stoppested. Funktionerne modtager et koordinat-sæt for bussen, ID'et for bussens tætteste rutepunkt, samt ID'et på stoppestedet. Bemærk at disse ID'er er hentet fra "BusRoute_RoutePoint"tabellerne og symboliserer derfor rutepunktet og stoppestedets placering på ruten, og ikke deres egentlige ID'er i henholdsvis "RoutePoint"og "BusStop"tabellerne. Funktionerne er ikke meget forskellige, ud over hvilken ChosenRoute tabel fra forrige funktioner, der tages i brug.

Herudover itereres der også i omvendt rækkefølge. Der vises kodeudsnit et for "CalcRouteLengthAsc", hvorefter funktionen forklares i detajler. Til sidst forklares forskellene mellem "CalcRouteLengthAsc og "CalcRouteLengthDesc"mere detaljeret. På kodeudsnit 8 kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik.

Kodeudsnit 8: CalcRouteLengthAsc. Udregner afstanden fra bus til stoppested

```
1 drop function if exists CalcRouteLengthAsc $$
2 create function CalcRouteLengthAsc(bus_pos_lon decimal(20,15), ↩
    bus_pos_lat decimal(20,15), BusClosestEndPointID int, ↩
    busStopId int)
3 returns float
4 BEGIN
5 declare RouteCounter int default BusClosestEndPointID;
6
7 select bus_lon from ChosenRouteAsc where id = RouteCounter into ↩
    R2x;
8 select bus_lat from ChosenRouteAsc where id = RouteCounter into ↩
    R2y;
9 set BusToStop = Haversine(R2y, bus_pos_lat, R2x, bus_pos_lon);
10
11 while RouteCounter < busStopId do
12   select bus_lon from ChosenRouteAsc where id = RouteCounter into↩
       R1x;
13   select bus_lat from ChosenRouteAsc where id = RouteCounter into↩
       R1y;
14   select bus_lon from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2x;
15   select bus_lat from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2y;
16   set BusToStop = BusToStop + Haversine(R2y, R1y, R1x, R2x);
17   set RouteCounter = RouteCounter+1;
18 end while;
19 drop temporary table ChosenRouteAsc;
20 return BusToStop;
21 END$$
```

"RouteCounter"initialiseres til det tætteste rutepunkt, hvorefter ID'et for dette punkt bruges til at hente det første koordinatsæt ud fra "ChosenRouteAsc". Dette koordinat sæt bruges sammen med bussens koordinater til at udregne afstanden fra bussen til rutepunktet. Denne udregning sker i den anden matematik funktion, "Haversine". Funktionen

vil ikke beskrives videre i dette afsnit, men for mere information henvises der til afsnittet *8.2.5: Komponent 5: Anvendt Matematik*. Herefter itereres der igennem ruten, hvor ID'et for det tætteste rutepunkt på bussen er den nedre grænse, og ID'et for stoppestedet er den øvre. Ved hver iteration findes afstanden mellem det nuværende punkt og det næste, og den totale afstand inkrementeres med denne værdi. Til sidst returneres den totale afstand. "CalcRouteLengthDesc"gøres der brug af "ChosenRouteDesc"tabellen i stedet for "ChosenRouteAsc". Desuden bruges ID'et for stoppestedet nu som den nedre grænse og det tætteste rutepunkt som den øvre, og R̈outeCounter"dekrementeres i stedet for inkrementeres. Dette kan ses på kodeudsnit 9

Kodeudsnit 9: CalcRouteLengthDesc forskel

```
1 ...
2 while RouteCounter > busStopId do
3   select bus_lon from ChosenRouteDesc where id = RouteCounter ↩
       into R1x;
4   select bus_lat from ChosenRouteDesc where id = RouteCounter ↩
       into R1y;
5   select bus_lon from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2x;
6   select bus_lat from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2y;
7 ...
8   set RouteCounter = RouteCounter-1;
9 ...
```

CalcBusAvgSpeed

Denne funktion tages i brug i slutningen af "CalcBusToStopTime"proceduren, og bruges til at udregne bussens gennemsnitshastighed. Dette bruges sammen med bussens afstand til stoppestedet, til at udregne, bussens tid til ankomst ved stoppestedet.

Da det i denne funktion er ligegyldigt, hvilken rute bussen kører på, er det også ligegyldigt hvilken vej den kører. Derfor er det kun nødvendigt at have en funktion til at udregne gennemsnitshastigheden.På kodeudsnit 10 kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik. Efter udsnittet forklares funktion detaljeret.

Kodeudsnit 10: CalcBusAvgSpeed. Udregner gennemsnitshastigheden for en bus.

```
1   create function CalcBusAvgSpeed(BusId int)
2 returns float
3 begin
4   drop temporary table if exists BusGPS;
5   create TEMPORARY table if not exists BusGPS(
6   id int auto_increment primary key,
7   pos_lat decimal(20,15),
8   pos_lon decimal(20,15),
9   busUpdateTime time
10 );
11
12 insert into BusGPS (pos_lat, pos_lon, busUpdateTime)
13 select GPSPosition.Latitude,  GPSPosition.Longitude, GPSPosition.↩
     Updatetime from GPSPosition
14 where GPSPosition.fk_Bus=BusId order by GPSPosition.ID asc;;
15
16 select count(id) from BusGPS into MaxPosCounter;
17 while PosCounter < MaxPosCounter do
18
19   select pos_lon from BusGPS where id= PosCounter into R1x;
20   select pos_lat from BusGPS where id= PosCounter into R1y;
21   select pos_lon from BusGPS where id = PosCounter+1 into R2x;
22   select pos_lat from BusGPS where id = PosCounter+1 into R2y;
23
24   set Distance = Distance + Haversine(R2y, R1y, R1x, R2x);
25   select busUpdateTime from BusGPS where id= PosCounter  into ↩
       ThisTime;
26   select busUpdateTime from BusGPS where id = PosCounter+1  into ↩
       NextTime;
27   set secondsDriven = secondsDriven + (Time_To_Sec(NextTime) - ↩
       Time_To_Sec(ThisTime));
28   set PosCounter = PosCounter + 1;
29 end while;
30 set speed = Distance/secondsDriven;
31 drop temporary table BusGPS;
32 return speed;
33 end $$
```

Først udhentes alle GPS positioner og opdateringstiderne for disse, for det relevante bus ID. Dette data indskrives i "BusGPS", en temporary tabel, med et ID, som autoinkrementeres fra 1. Antallet af GPS opdateringer fundet for den givne bus, bruges i en while-løkke som den øvre grænse. En counter instantieres til 1, og bruges som den nedre grænse. Ved hver iteration hentes den opdatering af bussens position, vis ID i "BusGPS"svarer

til counteren. Den næste opdatering i rækken hentes også ud, og afstanden mellem de to punkter udregnes ved hjælp af "Haversine"funktionen. For mere information om "Haversine", se afnsit *8.2.5: Komponent 5: Anvendt Matematik*. Den totale afstand inkrementeres med den udregnede afstand. Herefter findes opdateringstiden for det første punkt, samt opdateringstiden for det næste. De to tidspunkter omregnes til sekunder, og tiden for det først punkt trækkes fra tiden for det næste. Således findes den tid, det har taget bussen at køre det linjestykke, som spændes over de to punkter. Den totale tid inkrementeres med den fundende tid.
Efter fuldent gennemiterering af bussens positioner, divideres den total afstand med tiden det har taget at køre afstanden. Således findes gennemsnitshastigheden, og denne værdi returneres.

Haversine og CalcRouteLineDist

Disse to funktioner vil ikke vises som kodeudsnit, da de blot er MySQL implementeringer af matematiske funktioner.
Haversine bruges til at udregne afstanden mellem to punkter, i en fugleflugt. CalcRouteLineDist bruges til at udregne afstanden fra et punkt, til det nærmeste punkt på en linje, udspændt af to andre punkter. Udregninger vil blive vist og forklaret nærmere under afsnittet *8.2.5: Komponent 5: Anvendt Matematik* under henholdsvist "Haversine"og "Tætteste punkt på en linje".