Tallinna Ülikool
Digitehnoloogiate instituut
INFORMAATIKA ÕPPEKAVA

Ühistranspordi sõiduplaani andmete optimeerimine nutiseadmetele

Bakalaureuse töö

Autor: Taavi Tetlov

Juhendaja: Priidu Paomets

Autor: ..... „ ..... „2017

Juhendaja: ..... „ ..... „2017

Instituudi direktor: ..... „ ..... „2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	*(autor)*

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Taavi Tetlov (sünnikuupäev: 17.08.1993)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ühistranspordi sõiduplaani andmete optimeerimine nutiseadmetele”, mille juhendaja on Priidu Paomets, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinn, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Mõistete ja lühendite loetelu

General Transit Feed Specification (GTFS)	Ühistranspordiregistri avaandmete spetsifikatsioon
ÜTRIS	Ühistranspordi Infosüsteem
ZIP	Üks levinud kadudeta andmete pakkimise meetodeid ja vastav failivorming (AKIT).
Comma-separated values (CSV)	Komaeraldusega väärtused. Porditav failivorming, kus andmebaasikirjed on üksteisest eraldatud komadega. Selles vormingus on iga rida üks kirje, mille väljad on üksteisest komadega eraldatud. Komade järel võib olla suvaline arv tühikuid ja/või tabeldusmärke (*tab character*), sest neid ignoreeritakse. Kui väli ise sisaldab koma, siis peab kogu väli olema ümbritsetud jutumärkidega (e-Teatmik).
Encoding	Kodeerimine. Andmete või algoritmide teisendamine mingisse vajalikku vormingusse, sh koodi abil (salastust taotlemata) (AKIT).
Trip	Reis.
Itinerary	Reisiplaan, teekond.
Integer	Täisarv, mis on esitatav naturaalarvude vahena, st ..., -3, -2, -1, 0, 1, 2, 3, ...; laiendab naturaalarvude hulka nulli ja negatiivsete arvudega (AKIT).
Real	Arv, mille saab püsialusega arvusüsteemis esitada lõpliku või lõpmatu diskreetesitusega (AKIT).
Text	Andmebaasides kasutatav andmetüüp, mille

	pikkus on määratud vastavalt sõne pikkusele (ntext, text, and image (Transact-SQL)).
Blob	Suur kahendobjekt, bloob. Andmebaasi salvestatud suur bitiplokk (pilt, helifail, videoklipp, mõnikord ka binaarkood), mida andmebaasihaldur ise ei interpreteeri (e-Teatmik).
Base64	Kodeerimismeetod, mis teisendab binaarandmeid ASCII tekstiks ja vastupidi (e-teatmik).
B-Tree	*B-Tree* on balansseeritud puu, mis on ideaalne olukordadeks, kus puud hoitakse osalisel või täielikul kujul sisemälus. Seda kasutatakse mitmeteks funktsioonideks nagu otsing, eelkäija ning järglase leidmine, miinimum ning maksimum väärtuste leidmine, andmete sisestamine ning kustutamine (Neubauer).
Kilobyte (KB)	Kilobait. Ümmarguselt 1000 baiti, täpsemalt 2^10 ehk 1024 baiti. 1 bait on vajalik ühe märgi või sümboli edastamiseks (e-Teatmik).
Megabyte (MB)	Megabait. Ligikaudu 1 miljon baiti, täpsemalt 2^20 ehk 1 048 576 baiti või 1024 kilobaiti (e-Teatmik).
Gigabyte (GB)	Gigabait. 1 gigabait on ligikaudu 1 miljard baiti, täpsemalt 2^30=1 073 741 824 baiti (e-Teatmik).

Sissejuhatus

Tänasel päeval on mobiilirakendused väga populaarsed. Sama võib öelda ka ühistranspordi sõiduplaane kuvavate mobiilirakenduste kohta – ainuüksi Tallinna jaoks on loodud mitmeid rakendusi. Selline nähtus on võimalik tänu GTFS'i kasutusele tulekuga 2005 aastal, mis standardiseeris sõidugraafiku andmete spetsifikatsiooni ning populariseeris nende avaliku jagamise.

Antud töö eesmärk on jätkata GTFS andmebaasi optimeerimist autori enda mobiilirakendusele[1]. Optimeerimine koosneb andmestruktuuri muutmisest, indekseerimisest ja muudest nippidest, mille tulemusel peaks säilima või parenema päringute kiirus ja vähenema andmebaasi suurus. Samas ei ole mõistlik suurendada uue andmebaasi genereerimise keerukust kui tegelik kasu on väike. Pigem tuleb hoida süsteem võimalikult lihtne ja efektiivne.

Iseenesest ei ole GTFS'i andmebaasi suurus probleemiks kui andmebaas asub serveris. ÜTRIS'e poolt väljastatud Eesti ühistranspordi andmed võtavad ruumi umbes 104MB CSV formaadis. Tavaliselt pakuvad mobiilirakendused mingi kindla regiooni andmeid. Sellisel juhul on andmete maht väiksem ja käsitletav ka nutiseadmetele. Andmebaasi suurus ei tohiks moodsamatele nutiseadmetele probleemiks olla, kuna tänapäeval on sisemälud suure mahutavusega – on isegi rakendusi, mis vajavad 1GB mälu. Sellegipoolest oleks hea arvestada ka vanemate nutiseadmetega.

Autor märkas, et kõige suurem probleem on hoopis andmebaasi salvestamine ja uuendamine nutiseadmetel. Kompresseeritud andmebaasi, mille suurus on 5MB – selle allalaadimine ja lahti pakkimine on nutiseadmel aeganõudev protsess. Autor üritas selle probleemi lahendamiseks internetis leida lahendusi, kuid sellel teemal on vähe materjali – leida oli võimalik ainult üks põhjalikum töö. Peale selle ei olnud teisedki rakendused selles osas midagi ette võtnud. Antud töö kirjutamise hetkel on mobiilirakendus Trafi[2] ka teinud oma andmebaasi väiksemaks, kuid kahjuks ei ole võimalik uurida kuidas seda on optimeeritud.

Töö jaguneb kahte ossa. Esimesed kolm peatükki annavad teoreetilise aluse teemades nagu GTFS, SQLite ja andmebaasi optimeerimine. Teises pooles rakendatakse neid teadmisi. Esmalt uuritakse T.Szincsak'i ja A.Vagner'i loodud andmemudelit. Siis kirjeldab autor oma andmemudeli normaliseerimise lahendust ja töös kasutatud optimeerimise nippe.

1 Mobiilirakendus Tallinna Sõiduplaanid Google Play's https://play.google.com/store/apps/details?id=com.appwolves.tallinntransport
2 Mobiilirakendus Trafi Google Plays: https://play.google.com/store/apps/details?id=com.trafi.android.tr

1 Ühistranspordi voo spetsifikatsioon

Ühistranspordi voo spetsifikatsioon (GTFS) on TriMet ja Google koostöös välja töötatud standardne ühistranspordi sõidugraafikute talletamise spetsifikatsioon. Tänasel päeval kasutatakse seda laialdaselt üle maailma ja samuti kasutab seda ka Eesti Ühistranspordi Infosüsteem (ÜTRIS) oma andmete avalikustamiseks (Ühistranspordi infosüsteem).

1.1 Loomislugu

Enne ühistranspordi andmete ühtlustamist oli igal ühistranspordi teenusepakkujal oma viis andmete talletamiseks. See tähendas, et nende andmete erinevatel rakendustel kasutamiseks võis olla vajalik andmete ümbervormindamine.

Paljudel teenusepakkujatel olid oma veebilehed informatsiooni avalikustamiseks, mida kasutati lisatulu teenimiseks Google AdSense reklaamidega. Sellepärast ei soovitud teha andmeid kättesaadavaks teistele arendajatele, kes pakuksid konkurentsi ning ohustaksid külastatavust nende endi veebilehtedel (Roush, 2012).

GTFS'i idee sai alguse aastal 2005 TriMeti töötajast Bibiana McHugh'ist, kes tegeles igapäevaselt ühistranspordi andmetega. Paljudel agentuuridel, nagu ka TriMet'is, olid oma ühistranspordi reisiplaneerijad, kuid tihtipeale ei osanud tavaline inimene neid üles leida, eriti kui nad olid sattunud võõrasse ühistranspordi süsteemi.

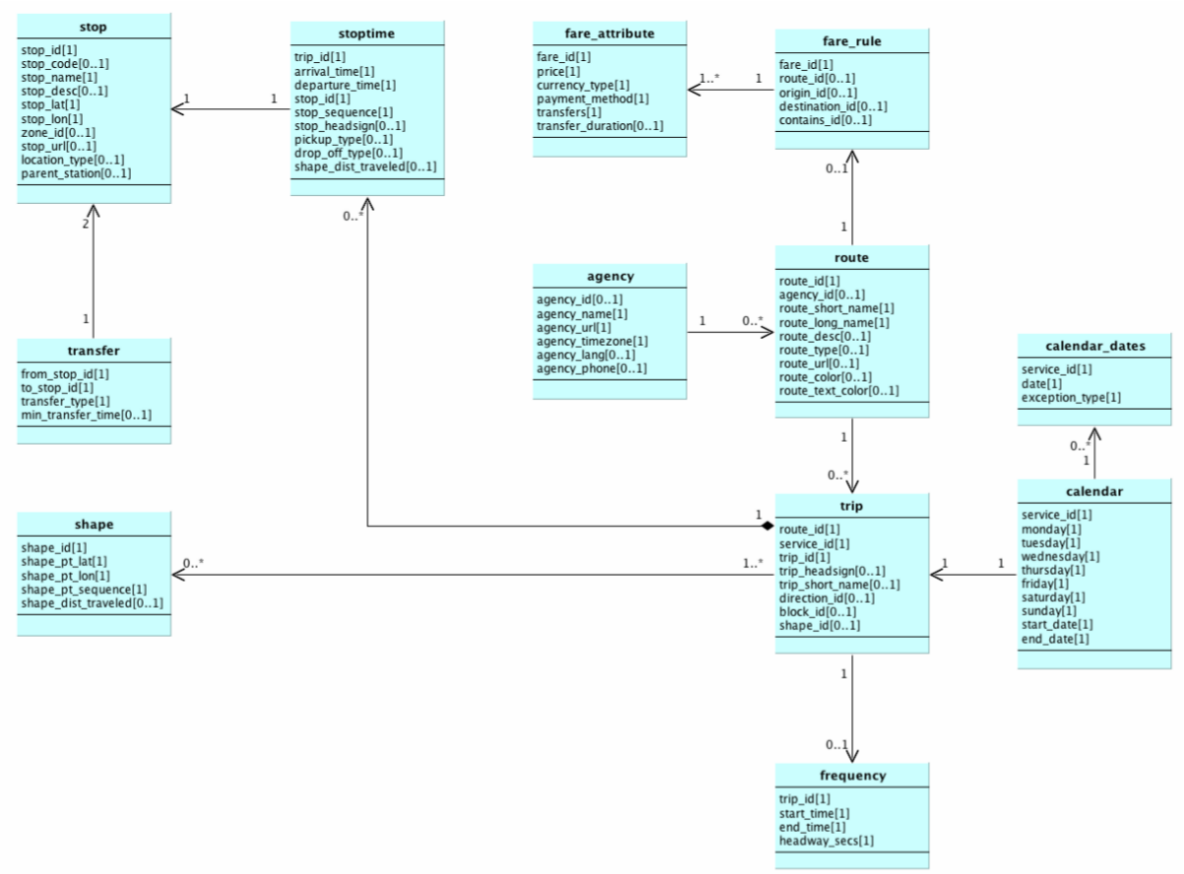
Peale edutuid partneri otsinguid tutvustati talle Google tarkvaraarendajat Chis Harrelson'i, kellel sarnane idee oli mõttes olnud, kuid polnud partnerit, kes pakuks vajalikke andmeid. Detsembris 2005 avalikustati Google Transit rakendus koos Portlandi linna ühistranspordi andmetega ja saadi väga positiivse tagasikaja (McHugh).

Spetsifikatsioon avaldati esialgu kui *Google Transit Feed Specification*. Kuna saavutati nii lai kasutajaskond, siis hiljem asendati *„Google"* eesliide *„General"* eesliitega (Hughes, 2009).

1.2 Andmestruktuur

GTFS'i voogu hoitakse ZIP failis, mis kompresseerib kokku mitmed tekstfailid. Iga tekstfail on analoog ühele tabelile, mis koondab enda alla mingit spetsiifilist tüüpi informatsiooni: peatused, marsruudid, peatumiste ajad jne. Andmed sisestatakse CSV formaadis, kus esimesel real on tabeli tulpade nimetused ja ülejäänud read on andmed (File Requirements).

GTFS'i andmemudel (vt Joonis 1) koosneb 14 omavahel seoses olevast tabelist (vt Tabel 1), millest 6 on kohustuslikud, ja lisaks valikuline *feed_info* tabel, mis annab infot voo pakkuja kohta ja ei ole seotud teiste tabelitega.



Joonis 1: GTFS andmemudel (GTFS Static Overview)

Tabel 1. Ülevaade GTFS tabelitest (GTFS Static Overview)

Failinimi	Nõutud	Kirjeldus
agency.txt	X	Transpordi teenusepakkujad. Näiteks Tallinnas on selleks Tallinna Linnatranspordi AS. Välja saab tuua igasugust lisainformatsiooni nagu kodulehekülg või telefoni number.
stops.txt	X	Peatuste andmed, kus ühissõidukid peatuvad reisijate peale võtmiseks ja maha panemiseks.
routes.txt	X	Ühistranspordiliinid. Näiteks bussiliinid 1A, 2, 3 jne.
trips.txt	X	Reisid iga liini kohta, mis koosnevad vähemalt kahe peatuse läbimisest mingi aja jooksul. Reis oleks näiteks 1A teekond Viru Keskus 5 - Viimsi keskus kell 5:30 – 5:55.
stop_times.txt	X	Kellaajad iga reisi kohta, mis kell transpordivahend peatusesse jõuab ja sealt väljub.
calendar.txt	X	Reiside teostamise ajakava nädalase plaani järgi esmaspäevast pühapäevani. Iga nädalapäeva kohta on eraldi kahendmuutuja väärtus - kas sel päeval transpordivahend sõidab või mitte. Samuti on kuupäevaliselt kirjas, mis ajavahemikus reise teostatakse - see annab võimaluse eraldi kirjeldada aastavahetusel sõitavat liini või siis mingil kuupäeval aeguvaid või/ja algavaid liinireiside plaane.
calendar_dates.txt		Erandid *calendar* tabeli suhtes, mida on vajaduse korral võimalik kasutada ka *calendar* tabeli asemel. Erandid sisestatakse üksikute kuupäevade kohta ja lisatakse kahendväärtus - kas konkreetsel kuupäeval liin sõidab või mitte. ÜTRIS kasutab seda tabelit enamasti riigipühade ülesmärkimiseks.
fare_attributes.txt		Transpordiliinipiletite hinnad.
fare_rules.txt		Ühendab transpordiliinid *routes* tabelis pileti hindadega *fare_attributes* tabelis.
shapes.txt		Koordinaadid, et visualiseerida kaardi peal liini marsruutide teekonnad.

frequencies.txt		Sisaldab ajavahemikke iga peatuse vahel, kui seda kasutada siis ei ole vaja kirja panna transpordi peatusesse jõudmise ja väljumise aega. ÜTRIS seda tabelit ei kasuta.
transfers.txt		Lisainfo kahe peatuse vahe läbimiseks. ÜTRIS seda tabelit ei kasuta.
feed_info.txt		Info GTFS voo väljastaja kohta.

2 SQLite ülevaade

SQLite andmebaasi mootor on arvatavasti kõige laialdasemalt kasutatud andmebaase. Täpset statistikat ei ole võimalik koguda kuna neid kasutatakse väga paljudes rakendustes ja seadmetes, mille üle pole võimalik arvet hoida, nagu näiteks nutitelefonid, operatsioonisüsteemid, veebilehitsejad, enamus televiisoreid, iTunes, PHP, Python jm (Most Deployed). Teda eristab tavapärasest andmebaasist see, et ta ei ole klient-serveri põhine ja tal ei ole eraldi serveri protsessi. Terve SQL andmebaas on ühes failis, millele tehakse lugemis ja kirjutamis päringuid (About SQLite).

SQLite üks suur erinevus esineb andmetüüpide deklareerimises. Enamus SQL andmebaasides määratakse andmetüüp tabeli veerule ja sinna ei ole võimalik muud tüüpi andmeid sisestada. SQLite on selles osas paindlikum - andmetüüp on määratud väärtusele endale, mitte tema konteinerile ehk veerule. See tähendab, et SQLite juhindub tabeliveerule määratud andmetüübist, kuid sisestada on võimalik ükskõik millist tüüpi andmeid. Kui veerutüüp on *integer,* siis sinna on võimalik sisestada ka sõnesid. Kuid numbrilise sisuga sõned nagu näiteks „123”, sisestatakse andmebaasi siiski täisarvulise väärtusena.

SQLite’is erisatakse hoiustamisklasse (ingl *storage class*) ja andmetüüpe (ingl *datatype*). Arendajatele antakse kasutada 5 klassi: NULL, INTEGER, REAL, TEXT ja BLOB. Näiteks hoiustamisklass INTEGER sisaldab endas 6 erineva pikkusega täisarvu tüüpi: 1, 2, 3, 4, 6 või 8 baiti. Kuna igale väljale saab määrata erineva andmetüübi, siis määratakse talle kõige parem andmetüüp (Datatypes In SQLite Version 3).

Huvitav tähelepanek on, et kõikidele tabeli ridadele on vaikimisi määratud unikaalne võti *rowid* (v.a vastava erandiga deklareeritud tabelid). Andmed *rowid’*ga tabelites on salvestatud *B-Tree* struktuuri kujul, kus *rowid* on võtmeks. See teeb rea otsimise *rowid* järgi on umbes kaks korda kiiremaks kui primaarvõtme või indekseeritud väärtuse järgi.

Kui *rowid* tabelis on deklareeritud primaarvõti täisarvuna, siis sellest veerust saab *rowid* alias. Selleks tuleb kasutada ühte nendest tabeli deklareerimise viisidest (SQL As understood By SQLite):

- CREATE TABLE t(x INTEGER PRIMARY KEY ASC, y, z);
- CREATE TABLE t(x INTEGER, y, z, PRIMARY KEY(x ASC));
- CREATE TABLE t(x INTEGER, y, z, PRIMARY KEY(x DESC));

3 Andmebaaside optimeerimine

Andmebaasi optimeerimine on vajalik andmete kiireks kätte saamiseks. Mida suuremaks lähevad andmebaasid, seda kriitilisem on ka õige optimeerimine, kuna suure hulga andmete läbi töötamine on aja- ja ressursirohke tegevus. Kaks peamist optimeerimise meetodit on indekseerimine ning päringuplaani optimeerimine.

Optimeerides andmebaasi nutiseadme jaoks, tuleb meeles pidada ka, et nutiseadmete protsessorid on nõrgemad kui arvutitel. Seega nutiseadmetele andmebaasi optimeerides oleks vajalik tulemusi ka vastaval seadmel testida.

3.1 Indekseerimine

Indekseerimine on tähtis õigete andmeridade kiiremaks leidmiseks andmebaasist. Mida rohkem on ridu andmebaasis, seda paremini on märgata indeksite efektiivsust, kuna ilma indeksiteta on otsitavate väärtuste ülesleidmiseks vajalik läbi töödelda terve tabel. Samal ajal suudavad indeksid pakkuda koheselt vajalike väärtuste asukohad tabelis.

Indeksite jaoks luuakse eraldi andmestruktuur, mis hoiab endas tabeli indekseeritud väljade kohta koopiat. Mõnes mõttes on see liialdamine, kuna tabelit ise ei muudeta, vaid luuakse uus andmestruktuur, mis viitab algsele tabelile. Sellegipoolest on see vajalik, kuna andmed seatakse uues andmestruktuuris järjekorda ning tänu sellele on sealt otsimine nagu telefonikataloogis - kõik andmed on järjestatud ja vajalikke sissekannete positsiooni leidmine on lihtsustatud ning kiirem (Winand).

Indeksid tuleks paigutada tabeli väljadele, mille järgi andmeid otsitakse või luuakse seoseid teiste tabelitega. Samuti on indeksid kasulikud ka sorteerimisel, kuigi see ei mõjuta oluliselt funktsiooni kiirust, kasutatakse siiski vähem ajutist mälu. Kui kõik vajalikud andmed on indeksitest võimalik kätte saada, siis ei ole vaja enam tabelile päringut tehagi, kuid sellise optimeerimise tulemus ei ole nii drastiline kui otsinguteks või seosteks kasutatavate väljade indekseerimine (Query planning).

3.2 Päringu teostamise plaan

„EXPLAIN QUERY PLAN” on SQL käsk, mis tagastab kirjelduse kuidas andmebaasi mootor kavatseb päringu teostada (vt Joonis 2). Eesmärk on saada tagasisidet ja saadud informatsiooni põhjal parandada indekseerimist ning päringut kuni leitakse võimalikult hea päringuplaan.

Põhiliselt tuleb jälgida kas tabelile tehakse SCAN või SEARCH (Explain query plan):

- SCAN - tabeli kõik read on vaja läbi töödelda õigete vastete leidmiseks. Alampäringuga (ingl subquery) loodud tabelitele tehakse alati SCAN kuna nad on ajutised tabelid ja ei oma indekseid.
- SEARCH - tabelist otsitakse õiged read koheselt üles primaarvõtme või indeksite abil.

selectid	order	from	detail
1	0	0	SEARCH TABLE route_trips AS rt USING COVERING INDEX ind_route_trips (route_id=? AND direction_code=?)
1	1	1	SEARCH TABLE trips AS t USING COVERING INDEX ind_trips (route_trip_id=?)
0	0	1	SCAN SUBQUERY 1 AS x
0	1	0	SEARCH TABLE itinerary AS i USING COVERING INDEX ind_itinerary (itinerary_id=?)
0	2	2	SEARCH TABLE stops AS s USING INTEGER PRIMARY KEY (rowid=?)

Joonis 2: Peatuste vaate EXPLAIN QUERY PLAN

3.3 Andmebaasiga ühenduse loomine

Selleks, et teha päring andmebaasile on esmalt vaja luua ühendus. Võimalik on seda ise alustada käsuga BEGIN_TRANSACTION või vastasel juhul see luuakse ja sulgetakse automaatselt iga päringu jaoks eraldi, mis on ebaefektiive, kui on vaja teha mitmeid päringuid. Seepärast on soovitatav see ise luua ja pärast päringute töö lõppu ühendus sulgeda käsuga END_TRANSACTION, sest siis saab ühe ühenduse jooksul teha mitu päringut (Lyon, 2003).

4 Ühistranspordi andmebaasi optimeerimine

GTFS andmetes esineb väga palju duplikaatandmeid, see ei pruugi olla probleem kui andmeid päritakse serveritest, kuna duplikaatandmed ei mõjuta otseselt päringute kiirust, ja andmete maht ei ole serverite jaoks ka liiga suur. ÜTRIS'e voos võtavad terve Eesti ühistranspordi andmed lahti pakituna ruumi umbes 104 MB.

Probleem tekib siis kui on vaja andmeid talletada tavakasutajate nutiseadmetesse, et andmed oleksid kättesaadavad ka internetiühenduseta. Kui andmebaas on väga suur, siis see tähendab pikka allalaadimisaega, kulutatakse palju võrgumahtu, faili lahti pakkimine võtab kaua aega ja kasutatakse ära palju ruumi seadme sisemälul.

Optimeerimise eesmärk on saavutada olek, kus oleks võimalikult vähe duplikaatandmeid ja andmed oleksid võimalikult kompaktsed. Samas ei ole mõistlik teha muudatusi, mis lisavad juurde keerukust, kuid ei vähenda kuigi palju andmebaasi suurust.

Esmalt uuritakse T.Szincsak'i ja A.Vagner'i andmemudelit ja tehakse selle efektiivsuse hindamiseks ka osaline katse. Peatükki teises pooles kirjeldab autor kuidas ta optimeeris andmebaasi oma mobiilirakenduse jaoks.

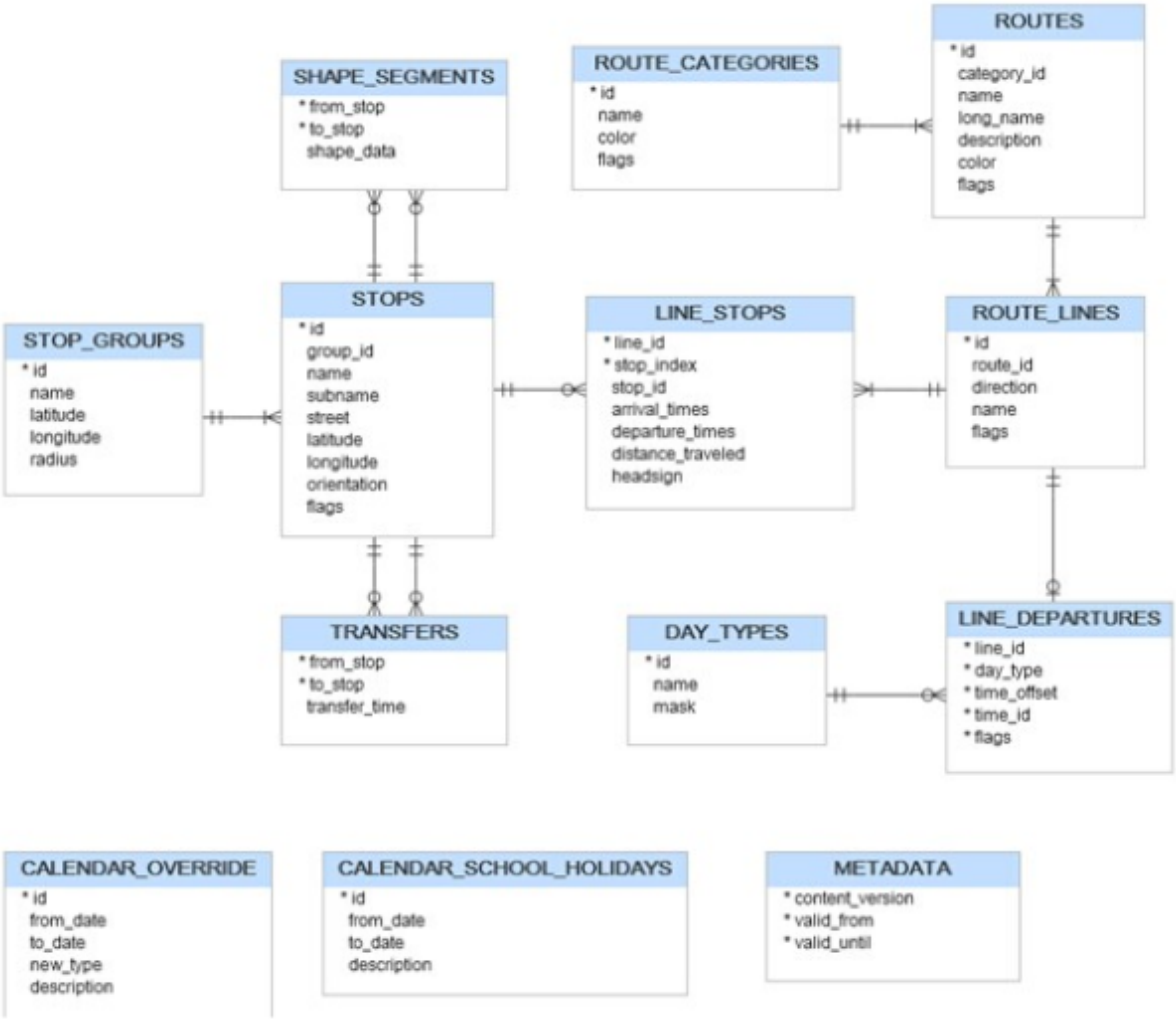
4.1 Szincsak'i ja Vagner'i andmemudel

GTFS'i andmemudeli muutmine ei ole kuigi levinud teema ja internetis on võimalik leida ainult üksikuid infokillukesi, mis seda teemat puudutavad. Autor leidis Tamas Szincsak'i ja Aniko Vagner'i poolt kirjutatud üsna põhjaliku töö *„Data Structure to Store GTFS Data Efficiently on Mobile Devices"*. See annab ülevaate kuidas on GTFS'i andmemudelit muudetud mobiilirakenduse „Budapest menetrend'i"[3] jaoks, et efektiivsemalt salvestada GTFS andmebaas lõppkasutajate nutiseadmetele. Järgnev peatükk võtab kokku selle töö põhiideed, mis seostuvad ka antud tööga.

Töös on väidetud, et üsna paljud mobiilsed rakendused vajavad andmete kättesaamiseks interneti ühendust ja andmete talletamine mobiilile ei ole populaarne. Selle põhjuseks on nutiseadmete mälu ja protsessori võimekuse limiteeritus. Tänaseks päevaks on aga Eesti ühistranspordi jaoks loodud mobiilirakendustes andmete salvestamine seadmetele üsna populaarne.

3 Mobiilirakendus Budapest Menetrend Google Play's https://play.google.com/store/apps/details?id=hu.donmade.menetrend.budapest

Efektiivsuse saavutamiseks kasutatakse objekt – relatsioonilist andmestruktuuri (vt Joonis 3). Erinevalt relatsioonilisest andmemudelist on mõndadesse väljadesse andmed sisestatud massiividena.



Joonis 3: Mobiilirakenduses Budapest Menetrend kasutatud andmemudel

Põhiline muudatus on tehtud *stoptimes* tabelile. Algselt on seal iga reisi peatumine eraldi kirja pandud. See tähendab, et kui ühel marsruudil tehakse 50 reisi, mis koosneb 20 peatumisest, siis selle kohta on tabelis 1000 rida andmeid. Selle asemel on nüüd kirja pandud marsruudi peatumiste jada üks kord ning iga peatuse saabumis ja väljumis ajad on grupeeritud massiivina vastavatele väljadele. Selline meetod lisab ainult 20 rida tabelisse, kus igasse ühte on ajad massiividena sisestatud. Selle jaoks on loodud uus tabel *line_stops*.

Juurde on lisatud *route_lines* tabel, mis talletab andmeid erinevate marsruutide kohta. Näiteks 1A bussiliinil on 3 marsruuti: Viru keskus 5 – Viimsi keskus, Viimsi keskus – Hobujaama ja

Hobujaama – Viimsi keskus. GTFS failis on need transpordiliini erinevused üles märgitud *trips* tabelis *direction_code* väljaga.

Mõndadel tabelitel on väli *flag*, mida kasutatakse erinevate andmete salvestamiseks ja iga tabeli juures võib ta omada erinevat tähendust. Näiteks *routes* tabelis võib see tähendada, et ühistransporti saab siseneda kõikidest ustest või reisi jaoks on vaja osta eraldi pilet. Tabelis *line_departure* võib märkida kas ühistransport on ratastooli sõbralik.

Autor viis ise läbi ka osalise katse andmemudeli efektiivsuse hindamiseks. Katse jaoks said loodud kõik vajalikud tabelid ja nendest ära täidetud *routes*, *route_lines* ja *line_stops*. Sellel hetkel oli autori enda andmebaasi suurus 4,6MB (ilma kellaaegade base64 kodeerimiseta, koos sekunditega („:00”) kellaaegade lõpus (vt Peatükk 5.2.3) ja indekseerimata). Esimeses katses sisestati *arrival_times* ja *departure_times* eraldi väljadesse nagu on ka selle admemudelil määratud - see andis tulemuseks umbes 7MB. Teises katses genereeriti *arrival_times* ja *departure_times* kokku üheks väljaks (vt Peatükk 5.2.3). See andis tulemuseks 4 196KB, kuid puudu oli veel *stops* ja *calendar* tabeli andmed.

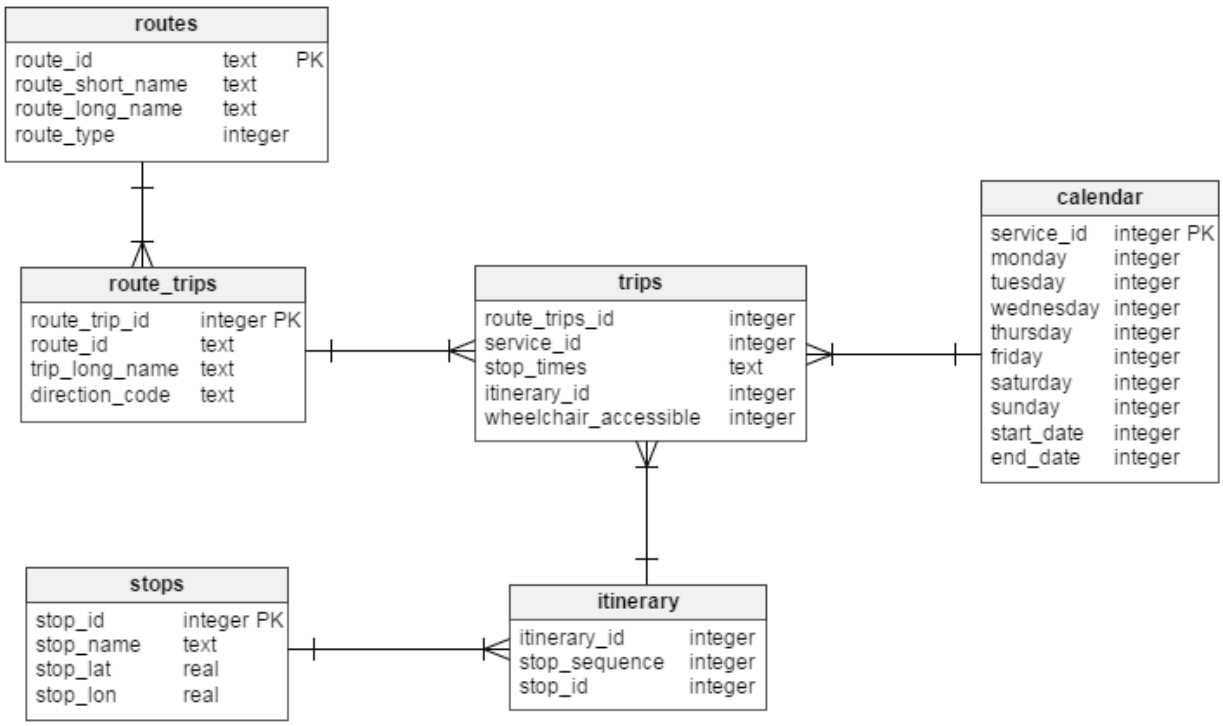
Autori jättis siin kohal katse pooleli kuna on näha, et andmebaaside suurused olid üksteisele üsna lähedal ning nende põhjal on juba võimalik mõningad järeldused teha. Samuti on *calendar* tabeli genereerimine keerukam protsess, kuna selle struktuur ja loogika ei sarnane enam algsele GTFS failile.

Nimelt luuakse uued *line_departure* andmeread marsruudi reiside põhjal *line_stops* tabelist. Pannakse kirja *line_id* ja *time_id*, mis on *line_departure* tabeli aegade massiivis reisi järjekorra number. Sellisel meetodil peaks andmete hulk olema võrdväärne reiside arvuga, ehk Tallinna andmete puhul peaks see tabel hoidma umbes 22 000 rida andmeid. Võrreldes esialgse umbes 1 000 reaga *calendar* tabelis, on andmete kasv 22 korda.

Autori enda andmebaas ilma *stops* ja *calendar* tabeliteta on 4 534KB. See teeb andmebaaside vaheks 338KB, mis on juba arvestatav suurus. Sellegi poolest võib *calendar* tabeli genereerimisest tulenev andmete mahu kasv muuta vahe pea olematuks.

4.2 Autori andmemudel

Tulemuste võrdlemiseks on andmed võetud 19.30.2017 veebilehelt http://www.peatus.ee/gtfs/. Voog sisaldab endas terve Eesti ühistranspordi andmeid, kust autor on Tallinna andmed välja filtreerinud. Järgnevad alampeatükid seletavad lahti kuidas autor andmemudelit optimeeris (vt Joonis 4).



Joonis 4: Autori loodud andmemudel

4.2.1 Tabel routes

Tabel *agency*'t ei ole vaja kasutada kuna Tallinna linna piires on ainult üks teenusepakkuja, seega *ageny_id* välja pole vaja ka kasutada *routes* tabelis. Välja *competent_auth* ei ole vaja kuskil kasutajatele kuvada, ega mobiilirakenduse süsteemseks tööks kasutada.

Väli *route_color* kirjeldab liinile iseloomulikku värvi kuueteistkümnendsüsteemis: bussid on rohelised, trollid sinised ja trammid punased. Kuid seda kasutatakse ka transpordiliigi täpsemaks klassifitseerimiseks. Selle vajadus ilmneb Harju maakonnas, kus kogu transport toimub ainult bussidega. Seal väli *route_type* viitab üldiselt bussiliinile, kuid selleks et teha vahet erinevat tüüpi bussiliinide vahel, kasutatakse *route_color* tabelit (Ühistranspordiregistri avaandmete spetsifikatsioon):

- DE2C42 – Avaliku teenindamise lepingu alusel teenindatav linnaliin, buss
- FF6319 – Kommertsalusel teenindatav linnaliin, buss
- F55ADC – Kommertsalusel teenindatav maakonnaliin, buss

- 00933C – Avaliku teenindamise lepingu alusel teenindatava maakonnaliin, buss
- E6FA32 – Kaugbussiliin (üldjuhul kommerts)

Autor leiab, et see on mitte-intuitiivne lähenemine ja võttis andmebaasis kasutusele oma *route_type* tüübid, kuna GTFS dokumentatsioonis ei ole kõikidele liinidele vastet. Avaliku linnaliini tüübina kasutatakse edasi numbrit „3” ja avaliku maakonnaliini saab dokumentatsiooni järgi asendada väärtusega „701”. Kommertsliinid ja kaugbussiliinid peab asendama ise valitud väärtustega.

Ruumi säästmiseks oleks võimalik ka *route_id* väärtused asendada, kuna algselt on nad esindatud 32 tähemärgise sõnena. Näiteks bussiliini 1A *route_id* on „62a774332c46f5b7d06d9c68214b3bbf”. Esiteks vajab pikem väärtus rohkem mälu ja kui kasutada täisarve, siis deklareerides need väljad *integer* tüübina, vajavad nad ka vähem ruumi iga tähemärgi kohta. Katsest tuli välja, et sellisel viisil väheneb andmebaas ainult 27KB. Autor on otsustanud selle lõplikust lahendusest välja jätta, kuna nii väike mahu kokkuhoid ei õigusta lisakeerukust.

4.2.2 Tabel trips

See on ridade arvult teine tabel, koosnedes 21 726’st reast. Väli *headsign* ei ole leidnud kuskil kasutust, seega on see jäänud tabelist välja. Välja *shape_id* on vaja juhul kui kasutada *shapes* tabelit.

Väga palju korduvaid andmeid esineb samade marsruutide kohta nagu *route_id*, *trip_long_name*, *direction_code* ja *shape_id*. Nende jaoks lõi autor uue tabeli *route_trips*, mis koondab enda alla marsruutide andmed. Uue tabeli primaarvõtmeks sai *route_trips_id*, mida kasutatakse seose loomiseks *trips* tabeliga. Uus tabel hoiab enda all nüüd 214 marsruuti, see on umbes 100 korda vähem duplikaatandmeid.

Sama on tehtud ka rakenduse „Budapest Menetrend” jaoks, kuid seal on tabelile antud nimeks *route_lines*. See tundub halva nime valikuna, kuna *routes* tähendab tõlkes marsruute ja *lines* transpordiliine. Autor arvab, et kõige õigem nimi ühistranspordi liini andmeid hoidvale *routes* tabelile oleks *lines* ja uuele marsruute hoidvale tabelile *routes*. Kuid segaduse vältimiseks jäi *routes* tabeli nimi samaks ja uus tabel sai nimeks *route_trips*, kuna ta on seos *routes* ja *trips* tabeli vahel.

4.2.3 Tabel stoptimes

Stoptimes tabel on kõige suurema mahuga, koosnedes 399 755 reast. Seetõttu on selle tabeli optimeerimine kõige tähtsam, kuna see võtab suurema osa andmebaasi mälust. Optimeerimine koosneb mitmest sammust ja lõpuks saab selle tabeli andmed panna *trips* tabelisse.

Tabel hoiab endas palju ridu - umbes 21 000 reisi kohta on eraldi veel kirja pandud iga peatumise identifikaator, saabumis ja väljumis aeg. Esimese sammuna koondatakse iga reisi *arrival_time* ja *departure_time* väljad kokku *stop_times* välja. Kuna *departure_time* ja *arrival_time* on tavaliselt sama väärtusega, siis piisab ühe väärtuse kirja panemisest. Erandjuhtudel sisestatakse mõlemad väärtused ja eraldatakse kaldkriipsuga

Peale neid muudatusi puudub vajadus *stop_sequence* välja jaoks, kuna *stop_times* massiivis on kõik andmed järjekorras. Samuti ei kasutata *pickup_type* ja *drop_off_type* väljasid, kuna neid ei näidata mobiilirakenduses.

Lõpuks jäi järgi 1:1 suhe Trips tabeliga ja järele jäänud andmed pannakse *trips* tabelisse. Tänu sellele on võimalik ruumi kokku hoida umbes 21 000 *trip_id* välja ja *stoptimes* tabeli indekseerimise pealt.

Lisaks on kellaajad võimalik muuta ruumiefektiivsemale kujule. Peale eelnevaid muudatusi oli *stop_times*'i välja sisu sellisel kujul: „05:57:00/06:00:00,06:01:00,06:02:00,06:03:00,06:04:00,06:05:00,06:06:00,06:08:00,06:08:00,06:10:00,06:10:00,06:11:00,06:13:00,06:14:00,06:16:00,06:18:00,06:19:00,06:22:00".

Esiteks saab ära kaotada sekundid, kuna nende väärtused on alati „:00" ja ei oma mingit väärtuslikku infot. Teiseks on võimalik ära kaotada kellaaegadelt kõik koolonid - hiljem saab väärtuse lahti dekodeerida, kus esimene pool tähistab tundi ja teine minutit.

Kolmandaks on kõik numbrilised väärtused asendatud base64 (vt Lisa 1) funktsiooni baasil ühekohaliste tähtedega, kus tund ja minut on eraldi kodeerimise võtmed. Autor tegi ise enda base64 funktsiooni, kus võtmena kasutatakse kellaaja täisarvulist väärtust, mitte binaarset väärtus. Viimasel juhul andmete maht hoopis kasvaks.

Neljandaks on ära kaotatud kõik komad kellaaegade vahelt. Kõik see kokku aitab andmebaasi mahtu vähendada üle 3MB võrra. Näide lõpuks genereeritud väärtusest: „F5/GAGBGCGDGEGFGGGIGIGKGKGLGNGOGQGSGTGW".

4.2.4 Tabel itinerary

Nagu ka T. Szincsaki ja A. Vagneri töös oli välja pakutud, siis *stoptimes* tabelis korduvad iga reisi kohta peatuste andmed. Antud lahendus on üsna sarnane nende omale – iga marsruudi kohta on kirja pandud ainult üks teekond peatustega, kuid autor pani need andmed eraldi tabelisse.

Selle jaoks lõi autor uue tabeli *itinerary*, mis sisaldab endas *stoptimes* tabelist *stop_sequence* ja *stop_id* välju, samuti uut primaarvõtit *itinerary_id*. Selle tulemusel sai peaaegu 400 000 reast peatuste andmetest 4225 rida.

Põhimõtteliselt oleks võimalik *stop_id*'d panna ka massiividesse. Siis väheneks andmebaasi suurus 4225'lt realt 227'le, kuid see tähendaks, et *stop_id* seostamiseks *stops* tabeliga oleks vaja koodis massiivist peatused eraldada ja iga peatuse nime kohta eraldi päring luua. See aga pikendaks oluliselt päringute teostamise aega. Lisaks sellele ei oleks võimalik leida kõiki samast peatusest väljuvad transpordiliine.

4.2.5 Tabel stops

Stops tabelis ei ole vaja midagi ümber teha, kuid vaja läheb ainult mõndasid välju nagu *stop_id*, *stop_name* ja koordinaate, juhul kui on vaja näidata peatuste asukohti kaardi peal (antud andmebaasis on nad sisse arvestatud).

Muid andmeid pole andmebaasi vaja sisestada kuna neid mobiilirakenduses ei kasutata. Näiteks väli *alias* on peatuse rahvanimi, mis on lisaks määratud ametlikule nimetusele. Välja *zone_id* ei ole vaja, sest see on dubleeritud *stop_id*'ga. Lisaks on välja jäetud sellised väljad nagu *stop_code* (peatuse kood ÜTRIS'es), *zone_name* (tsooni nimetus sõidupileti hinna määramiseks) ja L-Est koridnaatsüsteemi järgi määratud peatuste kordinaadid *lest_x* ja *lest_y* (Ühistranspordiregistri avaandmete spetsifikatsioon).

4.2.6 Tabel calendar

Erinevalt paljudest teistest tabelitest, on *calendar* tabelist vaja kõiki andmeid ja säilitatakse originaal kuju.

Võimalik oleks kõik päevade väljad grupeerida üheks, mis teeks päringud kergemini loetavamaks. Midagi sarnast on tehtud ka T.Szincsaki ja A.Vagneri kirjutatud töös. Sel moel andmete hoiustamisel andmebaasi oleks ruumisääst 10KB. Sellepärast otsustas autor jätta päevad eraldi lahtritesse, et vajadusel oleks võimalik teha päringuid ka päevade kaupa.

Enamasti grupeerib autor oma päringutes päevade väljad kokku, et mobiilirakenduses oleks lihtne neile inimloetav nimi anda:

- 1111100 – tööpäevad
- 0000010 – laupäev
- 0000001 - pühapäev

Mingil põhjusel on *calendar* ja *calendar_dates* tabelites palju identseid andmeid erinevate *service_id*'e alla paigutatud. Siin oleks võimalik programmiga andmed üle käia ja korduvad andmestruktuurid seostada juba eelnevalt salvestatud *service_id*'ga. Kogu c*alendar* tabeli sisu võtab ruumi 27KB, seega ruumi oleks võimalik kokku hoida sellest vähem. Võimalik, et seda on mõistlik teha, kui kasutusel oleks ka *calendar_dates* tabel.

Tabelis 2 on toodud näide sarnastest andmetest. Võttes arvesse, et *calendar_dates* tabelis on ridu 12 korda rohkem, siis sel juhul oleks võimalik kokku hoida juba arvestatavas mahus andmeid. Kuid see tähendaks ka keerulisemat koodi, mis kontrolliks, et kahes tabelis oleksid samad andmed.

Tabel 2. Tabeli *calendar* ja *calendar_dates* korduvad andmed

Calendar	7507,0,0,0,0,0,1,0,20160901,20180312	7508,0,0,0,0,0,1,0,20160901,20180312
Calendar_dates	7507,20170414,2 7507,20170416,2 7507,20170501,2 7507,20170604,2 7507,20170623,2 7507,20170624,2 7507,20170820,2 7507,20171224,2 7507,20171225,2 7507,20171226,2 7507,20180101,2 7507,20180224,2	7508,20170414,2 7508,20170416,2 7508,20170501,2 7508,20170604,2 7508,20170623,2 7508,20170624,2 7508,20170820,2 7508,20171224,2 7508,20171225,2 7508,20171226,2 7508,20180101,2 7508,20180224,2

4.3 Optimeerimine

Esimese asjana said paika pandud tabelite primaarvõtmed. Sellistel tabelitel nagu *trips* ja *itinerary* ei olegi primaarvõtit, vajadusel on võimalik kasutada ka automaatselt loodud *rowid* välja.

Teiseks on lisatud indeksid tabeli väljadele, mille järgi otsitakse, sorteeritakse ning luuakse seoseid tabelite vahel (vt Tabel 3). Selles veendumiseks kasutas autor EXPLAIN QUERY PLAN käsku.

Tabel 3. Autori andemudeli indeksid

Indeksi nimi	**Tabel**	**Väljad**
ind_routes	routes	route_type
ind_route_trips	route_trips	route_id, direction_code
ind_trips	trips	route_trip_id, service_id, itinerary_id
ind_calendar	calendar	end_date
ind_itinerary	itinerary	itinerary_id, stop_id

Kolmandaks kirjutas autor funktsioonid, mis loovad ühenduse andmebaasiga selleks ajaks kuni kõik päringud lõpetatakse. Ühendus on määratud klassi globaalsele muutujale *db* (vt Koodinäide 1).

```
public static void startTransactions() {
  db = helper.getReadableDatabase();
  db.beginTransaction();
}
public static void endTransactions() {
  db.setTransactionSuccessful();
  db.endTransaction();
  db.close();
}
```

Koodinäide 1: Androidis andmebaasiga ühenduse loomine

5 Tulemused

Selles peatükis võrreldakse GTFS ja autori poolt muudetud andmebaase. Esiteks võrreldakse päringuid, veendumaks et muudetud andmebaasist on võimalik sama hästi andmeid välja pärida. Teiseks andmebaasi suuruseid ja kolmandaks reaalset andmebaasi allalaadimis kiirust mobiilsele seadmele.

5.1 Päringud

Päringud on koostatud nii, et neist saadud andmetega oleks võimalik autori mobiilirakenduses luua klassid, mille põhjal luuakse lõppkasutajatele vaated. Klasside loomisaeg on ka päringu aegadesse sisse arvestatud. Autor tegi mõõtmised kolme põhilise vaate kohta: liinid, liini peatused ja marsruudi sõidugraafikud (vt Tabel 4).

Keskmise tulemuse saavutamiseks jooksutati päringuid 5 korda (vt Lisa 2) Samsung S4 mini GT-I9195 nutitelefonis. Päringud tehti trammi liinile 2, sest tema andmed asuvad andmebaasi tabelites peaaegu lõpus. Sellisel viisil on võimalik kõige paremini näha indeksite efektiivsusest või puudulikust. Päringute tulemused analüüsitakse järgnevates alampeatükkides.

Tabel 4. Päringute keskmine teostamise aeg

	Liinid	Peatused	Sõidugraafikud
Originaal			
indekseerimata	302	383	524
indekseeritud	214	1	24
Autori andmebaas			
indekseerimata	332	188	80
indekseeritud	179	2	63
indekseeritud ilma base64	-	-	34

5.1.1 Liinid

Liinide vaate päring (vt Tabel 5) võib olla üsna aeglane kui vaates on palju liine. Konkreetses päringus on välja otsitud kõik bussiliinid, mida on Tallinna andmetes ka kõige rohkem. Kõikide transpordi liinide kättesaamiseks on iseenesest vaja ainult kasutada *routes* tabelit, kuid osad liinid võivad alustada või lõpetada teenindamist millalgi tulevikus või teenindada mingil kindlal perioodil nagu näiteks aastavahetustel. Sellepärast on vaja kontrollida *calendar* tabelist kas esineb erinevusi ühe liini *start_date* ja *end_date* väljadel. Selline päring on aga

üsna mahukas, eriti bussiliinide otsingus, kuna *calendar* tabeliga seose leidmiseks on vaja läbi analüüsida suurem osa *routes*, *route_trips*, *trips* ja *calendar* tabelite andmetest.

Selle parandamiseks oleks võimalik, kas *calendar* tabelis koondada sarnaste väärtustega read ühe *service_id* alla või genereerida *routes* tabelisse vastavad erandid.

Tabel 5. Liinide vaate päringud

GTFS	SELECT r.route_id, r.route_short_name, r.route_long_name, c.start_date, c.end_date FROM routes r JOIN trips t ON t.route_id = r.route_id JOIN calendar c ON t.service_id = c.service_id WHERE r.route_type = 3 GROUP BY r.route_short_name, c.end_date ORDER BY r.route_short_name*1
Autori andmemudel	SELECT r.route_id, r.route_short_name, r.route_long_name, c.start_date, c.end_date FROM routes r JOIN route_trips rt ON rt.route_id = r.route_id JOIN trips t ON t.route_trip_id = rt.route_trip_id JOIN calendar c ON c.service_id = t.service_id WHERE r.route_type = 3 GROUP BY r.route_short_name, c.end_date ORDER BY route_short_name*1

5.1.2 Peatused

Peatuste vaate jaoks on vaja leida üks marsruudi reis ja tema peatumiste jada (vt Tabel 6). Kuigi selle päringu sooritamiseks on ka vaja ühendada ka palju tabeleid, siis erandiks on see, et igas tabelis on vaja läbi analüüsida väike kogus andmeid, mis on indeksitega võimalik kiirelt üles leida.

Tabel 6. Liini peatuste vaate päringud

GTFS	SELECT st.stop_id, s.stop_name FROM StopTimes st JOIN (SELECT trip_id from Trips t WHERE t.route_id = "1b9660ba37d124940064403c88a1b65e" AND t.direction_code = "A>B" LIMIT 1) x ON x.trip_id = st.trip_id JOIN Stops s ON st.stop_id = s.stop_id
Autori andmemudel	SELECT i.stop_id, stop_name FROM itinerary i JOIN (SELECT itinerary_id FROM route_trips rt LEFT JOIN trips t ON t.route_trip_id = rt.route_trip_id WHERE rt.route_id ="1b9660ba37d124940064403c88a1b65e" AND rt.direction_code = "A>B" LIMIT 1) x ON x.itinerary_id = i.itinerary_id LEFT JOIN stops s ON s.stop_id = i.stop_id

5.1.3 Sõidugraafikud

Sõidugraafikute vaates on märgata üsna suur ajaline erinevus indekseerimata andmebaaside juures. See on sellepärast, et andmeridade arv on autori enda andmebaasis märgatavalt väiksem. Kuid indekseeritud andmebaasis on GTFS kiirem, kuna andmed on koheselt õigel kujul kättesaadavad ja ei vaja hilisemat töötlemist peale andmebaasist kätte saamist.

Autori enda andmebaas on indekseeritult umbes sama kiire, kuid päringu teeb aeglasemaks fakt, et andmed on vaja sõnest välja otsida ja base64 funktsiooniga lahti dekodeerida (vt Lisa 3). Base64 funktsioon lisab andmete kättesaamisele juurde umbes 29ms ja sõnest väärtuse eraldamine kuskil 10ms.

Autori andmebaasi päringus on võimalik ära kasutada *route_trip_id*, mille järgi on võimalik kohe õige marsruudi peatused üles leida (vt Tabel 7). Sellisel juhul tuleks see identifikaator

eelnevalt muutujana mällu salvestada liini peatuste vaate juures. Seal otsitakse välja kõik liini marsruudid, kust saaks küsida lisaks ühe *route_trip* identifikaatori.

T.Szincsaki ja A.Vagneri andmemudelis (vt Peatükk 5.2) on marsruudi peatuse kellaajad juba paigutatud ühte välja. See tähendab, et sarnane päring peaks tagastama ainult ühe rea andmeid. Vastunäitena saadakse praegu konkreetse näite puhul 361 rida, kus igast reast tuleb õigel postisioonil olev aeg välja võtta. Ühte rida tagastava päringu aeg oleks kiirem ja samuti ka massiivi tükeldamine eraldi aegadeks, kuid base64 funktsiooni aeg jääks samaks, kuna sealt käiks läbi ikka sama palju andmeid.

Tabel 7. Sõidugraafiku vaate päringud

GTFS	SELECT t.trip_id, st.departure_time, c.monday||c.tuesday||c.wednesday|| c.thursday||c.friday||c.saturday||c.sunday AS date_code FROM Trips t JOIN StopTimes st ON t.trip_id = st.trip_id JOIN Calendar c ON c.service_id = t.service_id WHERE t.route_id = "1b9660ba37d124940064403c88a1b65e" AND t.direction_code = "A>B" AND st.stop_sequence = "3" AND c.end_date = "20180318"
Autori andmemudel	SELECT t.rowid AS trip_id, t.stop_times, monday||tuesday||wednesday|| thursday||friday||saturday||sunday AS date_code FROM trips t JOIN calendar c ON c.service_id = t.service_id WHERE t.route_trip_id ="203" AND c.end_date = "20180318"

5.2 Andmebaaside suurused

Andmebaasi mahtude erinevused on välja toodud Tabelis 8. GTFS andmebaas sisaldab samu andmeid, mis autori loodud (vt Lisa 4). Lisaks on GTFS andmebaasi *stoptimes* tabeli aegadel ära võetud sekundid, kuna need on alati väärtusega „:00”, ei anna mingit väärtuslikku informatsiooni ja on üsna lihtne optimisatsioon. Voo kompresseerimiseks ZIP faili kasutatakse java.util.zip paketti.

Mahult on tähtis vaadata indekseeritud andmebaase, kuna need jõuavad reaalselt lõppkasutaja rakendusse. Võib tähele panna, et autori optimeeritud andmebaas vajab ka vähem ruumi indekseerimiseks, kuna andmeridu on tabelites vähem.

Uus andmebaas võtab lõppkasutaja seadmel 1,8MB, mis 19MB võrra vähem esialgsest, ja kompresseeritud kujul tuleb üle võrgu allalaadida 608KB, mis on 4,6MB vähem võrreldes GTFS andmebaasiga. See teeb rakenduse lõppkasutajatele andmebaasi allalaadimise ja lahti pakkimise kiiremaks. Täpsemalt on seda uuritud järgmises peatükkis.

Tabel 8. Andmebaaside suurused (KB)

	GTFS	Autori andmemudel
Andmebaasi suurus	12 999	1 365
Andmebaasi suurus (ZIP)	3 215	525
Andmebaasi suurus indeksitega	20 859	1 797
Andmebaasi suurus indeksitega (ZIP)	5 222	608

5.3 Andmebaasi salvestamise kiirus

Kõige tähtsam on andmemahu vähendamise juures asjaolu, et see teeb lõppkasutajale uute andmete salvestamise nutiseadmele kiiremaks. Funktsioon koosneb mitmest osast: andmete allalaadimine, lahti pakkimine ja kompreseeritud faili kustutamine.

Tabelis 9 toodud andmete põhjal on näha, et allalaadimise kiirus on vähenenud üle 5 korra – 11,4 sekundilt 2,1 sekundile ning kogu protsess on vähenenud 12 korda 44,9 sekundilt 3,7 sekundi peale. Tuleb märkida, et allalaadimis kiirus võib varieeruda ka vastavalt võrguühendusele ning lahti pakkimine vastavalt protsessori võimsusele.

Tabel 9. Andmebaasi salvestamisaeg nutiseadmele

Andmebaas	Allalaadimis kiirus (ms)	Kogu protsess (ms)
GTFS	11 412	44 890
Autori andmemudel	2 123	3 720

Kokkuvõte

Käesoleva töö eesmärgiks oli optimeerida GTFS andmemudelit nutiseadmetele, mida saaks kasutada autori enda mobiilirakenduse tööks. Suur osa tööst hõlmas andmemudeli normaliseerimist ja dubleeritud andmete vähendamist. Samuti tehti mitmeid ettepanekuid andmete esitamiseks ruumisäästlikumal kujul, kuid enamus nendest ei läinud kasutusse kuna ruumisääst polnud piisav, et lisatud keerukus ennast õigustaks. Samas peatumiste kellaajad said base64 funktsiooniga ära kodeeritud. Ehkki see suurendab andmete pärimise aega pea 30ms, vähenes andmebaasi suurus 4,6MB pealt 1,3MB'le.

Töö esimesed kolm peatükki annavad teoreetilise ülevaate teemadest nagu GTFS, SQLite ja andmebaasi optimeerimine. Praktilises pooles sai vaadatud teiste autorite poolt loodud andmestruktuuri. Osalise katse põhjal võis järeldada, et sealne andmebaasi suurus tuleb sarnane autori omale, juhul kui kasutada samu kellaaegade kodeerimise võtteid. Sealses töös olid samuti korduvate andmete jaoks tehtud eraldi tabelid ning kellaaegade hoiustamiseks kasutati massiive, kuid kellaajad grupeeriti marsruudi peatuste kaupa. Vastupidiselt grupeeris antud bakalaureusetöö autor kellaajad reiside kaupa. Esimese eelis on see, et sõidugraafikute näitamiseks tagastatakse ainult üks massiiv, kus on kõik andmed juba olemas. Autori enda andmebaasis on vaja tagastada kõik marsruudi reiside ajamassiivid ning neid võib olla päris palju - peatükkis 6.1.3 loodud päring tagastas 361 rida andmeid.

Töö käigus valmis autori enda andmemudel, mis sobiks kasutamiseks lihtsamate rakenduste jaoks, mis ainult kuvavad sõidugraafikute informatsiooni. Samuti on siit võimalik leida mõtteid, et täiendada mõnda muud juba olemas olevat andmemudelit nagu seda tegi ka autor ise katsetades Szincsak'i ja Vagner'i loodud andmemudeliga. Nimelt nende andmebaasi *arrival_time* ja *departure_time* genereeris antud bakalaureuse töö autor kokku üheks väljaks, selleks, et tulemusi oleks võimalik võrrelda autori enda andmebaasiga. Selline tehnika muutis nende andmebaasi 7MB pealt 4,2MB'le.

Tulemuste peatükkis oli näha, et andmebaasi suurus ja nutiseadmele salvestamise kiirus on tohutult paranenud. Andmebaasi maht vähenes 20,9MB pealt 1,8MB'le, kompresseeritult 5,2MB pealt 0,6MB peale. See aga mõjutas omakorda ka salvestamise aega, mis langes 44,9 sekundi pealt 3,7 sekundi peale.

Kasutatud kirjandus

e-Teatmik: IT ja sidetehnika seletav sõnaraamat. (kuupäev puudub). *Vallaste e-teatmik.* Loetud aadressil http://vallaste.ee/

AKIT - Andmekaitse ja infoturbe leksikon. (kuupäev puudub). *Cybernetica AS*. Loetud aadressil http://akit.cyber.ee/

ntext, text, and image (Transact-SQL). (2016, 30. juuli). *Microsoft Docs.* Loetud aadressil https://docs.microsoft.com/en-us/sql/t-sql/data-types/ntext-text-and-image-transact-sql

Neubauer, P. (kuupäev puudub). *B-Trees: Balanced Tree Data Structures.* Loetud 17.04.2017 aadressil http://www.bluerwhite.org/btree/

Ühistranspordi infosüsteem. (kuupäev puudub). *Maanteeamet.* Loetud 14. aprill 2017 aadressil https://www.mnt.ee/et/uhistransport/uhistranspordi-infosusteem

Roush, W. (2012, 21. veebruar). *How (and Why) the Search Giant is Remapping Public Transportation.* Loetud aadressil http://web1.ctaa.org/webmodules/webarticles/articlefiles/Spring_12_DigitalCT_Google_Transit.pdf

McHugh, B. (kuupäeva puudub). *Pioneering Open Data Standards: The GTFS Story.* Loetud 14.aprill 2017 aadressil http://beyondtransparency.org/chapters/part-2/pioneering-open-data-standards-the-gtfs-story/

Hughes, J. (2009, 19. oktoober). *proposal: remove "Google" from the name of GTFS.* Loetud aadressil https://groups.google.com/forum/#!msg/gtfs-changes/ob_7MIOvOxU/zEScjv6VLBMJ

File Requirements. (kuupäev puudub). *Google Developers.* Loetud 14. aprill 2017 aadressil https://developers.google.com/transit/gtfs/reference/file-requirements

GTFS Static Overview. (kuupäev puudub). *Google Developers*. Loetud 14. aprill 2017 aadressil https://developers.google.com/transit/gtfs/

Most Deployed. (kuupäev puudub). *SQLite.* Loetud 14. aprill 2017 aadressil https://sqlite.org/mostdeployed.html

About SQLite. (kuupäev puudub). *SQLite*. Loetud 14. aprill 2017 aadressil https://www.sqlite.org/about.html

Datatypes In SQLite Version 3. (kuupäev puudub). *SQLite*. Loetud 14. aprill 2017 aadressil https://www.sqlite.org/datatype3.html

SQL AS understood by SQLite. (kuupäev puudub). *SQLite*. Loetud 14. aprill 2017 aadressil https://www.sqlite.org/lang_createtable.html

Query Planning. (kuupäev puudub). *SQLite*. Loetud 14. aprill 2017 aadressil http://www.sqlite.org/queryplanner.html

Explain Query Plan. (kuupäev puudub). *SQLite*. Loetud 14. aprill 2017 aadressil https://www.sqlite.org/eqp.html

Lyon, J. (2003, 10. september). *SQLite Optimization FAQ*. Loetud aadressil http://codificar.com.br/blog/sqlite-optimization-faq/

Szincsak, T., Vagner, A. (2014, september). *Data Structure to Store GTFS Data Efficiently on Mobile Devices.* Loetud aadressil http://ebib.lib.unideb.hu/ebib/CorvinaWeb?action=cclfind&resultview=longlong&ccltext=idno+BIBFORM055846

Ühistransporditegistri avaandmete spetsifikatsioon. (kuupäev puudub). *Maanteeamet.* Loetud 14. aprill 2017 aadressil https://www.mnt.ee/sites/default/files/elfinder/article_files/uhistranspordiregistri_avaandmete_spec_v1_3.pdf

Extended GTFS Route Types. (kuupäev puudub). *Google Developers.* Loetud 14. aprill 2017 aadressil https://developers.google.com/transit/gtfs/reference/extended-route-types

Josefsson, S. (2006, oktoober). *The Base16, Base32, and Base64 Data Encodings*. Loetud aadressil https://tools.ietf.org/html/rfc4648

Winand, M. (kuupäev puudub). *Anatomy of an SQL Index.* Loetud 14.aprill 2017 aadressil http://use-the-index-luke.com/sql/anatomy

Summary

GTFS Data Optimization for Smart Devices

Bachelor's Thesis

The aim of this Bachelor's Thesis was to optimize GTFS data model for smart devices which could be used in author's mobile application. Most of the thesis is about normalization and reducing duplicated data entries. Also suggestions were made on representing some data on more efficient form but many of them were not used as they did not reduce database size significantly enough to justify extra complexity added to process. Nevertheless stoptimes were encoded with base64 function. Although it slows query almost 30ms, it reduces database size from 4,6MB to 1,3MB.

First half of Thesis gave a theoretical overview on topics as GTFS, SQLite, and database optimization. Practical section covered similar work by Szincsak and Vagner. Partial experiment on their data model revealed that the size of database will be similar if same encoding techniques are used. In their work duplicated data was also concentrated into other tables and arrays were used to store times which were grouped by route stops. On the contrary, author of this Thesis grouped times by trips. The advantage of the first one is that the database will return only one row for each timetable view. Author's database will return all time arrays of route trips and there might be a lot of them as query in chapter 6.1.3 returned 361 rows of data.

The product of this work was author's own optimized database which can be used for simpler mobile applications that display timetables of public transportation. Also this work can offer ideas that can be used to optimize some other database as author himself used some of them in experiment with Szincsak's and Vagner's data model. Specifically columns *arrival_time* and *departure_time* were put together into one field, so that results would be comparable with author's database. That optimization reduced their database from 7MB to 4,2MB.

In final result chapter we saw that size of database and downloading it to smart device has drastically improved. Size reduced from 20,9MB to 1,8MB and compressed database from 5,2MB to 0,6MB. These effect were visible in downloading process as it dropped from 44.9 seconds to 3,7 seconds.

LISAD

LISA 1. Base64 kodeerimine

Tabel 10. Base64 tähestik (S.Josefsson, 2006)

Table 1: The Base 64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

LISA 2. Päringute täitmisaeg

Tabel 11. Päringute täitmisele kulunud katseajad nutiseadmel Samsung S4 mini GT-I9195

	1	2	3	4	5	Keskmine
Liinide vaade						
originaal	324	295	295	298	299	302
originaal indektistega	211	228	236	209	190	214
vähendatud	359	331	312	330	329	332
vähendatud indektistega	201	149	181	176	186	179
Peatuste vaade						
originaal	374	377	381	412	373	383
originaal indektistega	3	1	1	1	1	1
vähendatud	224	199	161	159	195	188
vähendatud indektistega	2	2	2	2	1	2
Sõidugraafikute vaade						
originaal	522	519	521	526	533	524
originaal indektistega	40	20	23	19	19	24
vähendatud	120	86	59	71	63	80
vähendatud indektistega	61	55	57	58	84	63
vähendatud indeksitega, ilma base64	38	31	30	30	43	34

LISA 3. Stoptimes dekodeerimine

Stoptimes dekodeerimiseks loodi kaks funktsiooni, kus *getDepartureTime* otsib sõnest välja õige väärtuse ja *formatTime* teeb base64 dekodeerimise ning tagastab andmed algsel kujul (vt Koodinäide 2).

Muutuja timeMap on java HashMap klass, kus on kõik base64 väärtused kirjas. Võimalik on kasutada ka Stringi, kus kõik väärtused on õiges järjekorras, ning kasutada String klassi charAt funktsiooni, kuid selle tulemused olid peaaegu 10ms aeglasemad.

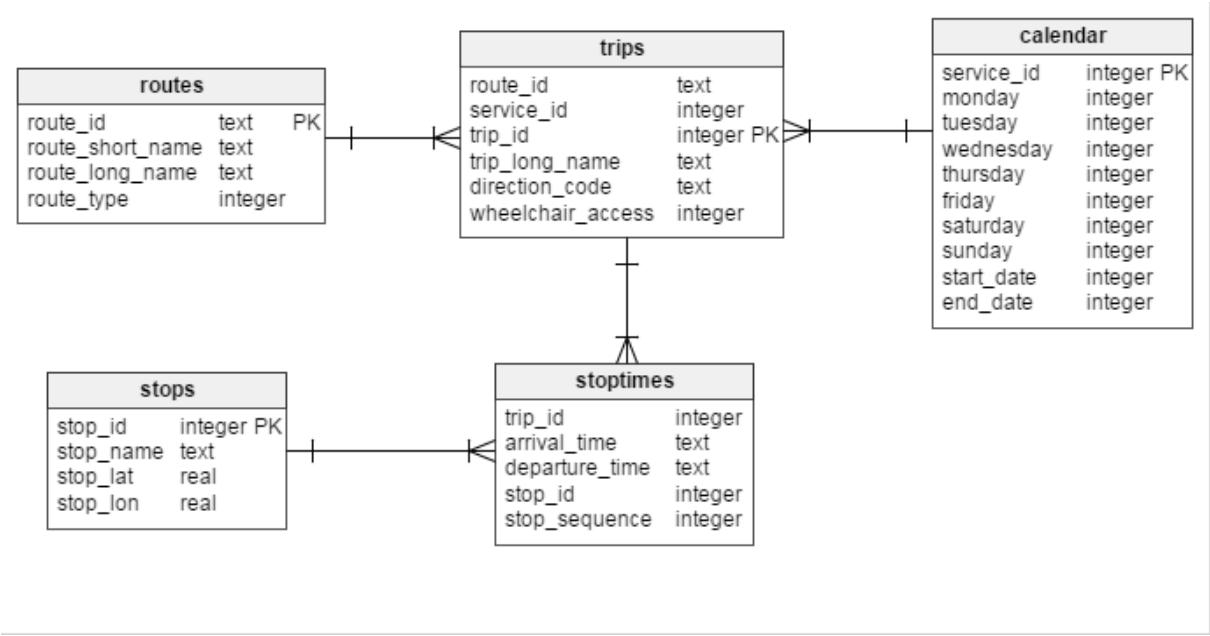
```
public static String getDepartureTime(String in, int pos) {
  int timeStartPos = pos*2;
  int timeEndPos = timeStartPos + 2;
  int departureTimeCounter = 0;
  int newTimeCount = 0;

  while ((newTimeCount = StringUtils.countMatches(in.substring(0, timeEndPos), "/")) != departureTimeCounter) {
    departureTimeCounter = newTimeCount;
    timeEndPos = (pos * 2) + (departureTimeCounter * 3) + 2;
  }
  timeStartPos = timeEndPos - 2;

  if (in.charAt(timeEndPos) == '/') {
    timeStartPos += 3;
    timeEndPos += 3;
  }

  return formatTime(in.substring(timeStartPos, timeEndPos));
}

public static String formatTime(String time) {
  String hour = timeMap.get(time.charAt(0));
  String minute = timeMap.get(time.charAt(1));
  return String.format("%02d", hour) + ":" + String.format("%02d", minute);
}
```

Koodinäide 2: Funktsioonid stoptimes dekodeerimiseks

LISA 4. Võrduseks kasutatud GTFS andmemudel

Tabel 11. Võrdluseks kasutatud GTFS andmebaasi indeksid

Indeksi nimi	Tabel	Väljad
ind_routes	routes	route_type
ind_trips	trips	route_id, service_id, direction_code
ind_calendar	calendar	end_date
ind_stoptimes	stoptimes	trip_id, stop_id, stop_sequence



Joonis 5: Võrdluseks kasutatud GTFS andmemudel vähendatud andmetega