



GTFS till postGIS

Nedladdning, bearbetning och visualisering av kollektivtrafikdata

Examensarbete
Mikael Leonidsson
Avesta Juni 2024



ABSTRACT

This thesis presents a method for managing and populating a PostgreSQL database with GTFS (General Transit Feed Specification) data. The aim is to make data available to various stakeholders within Region Dalarna, focusing on facilitating visualization and analysis in a Geographic Information System (GIS). A version control system has been implemented to enable comparisons with historical data.

The work involves programming in R to interact with the database, emphasizing creating a robust error handling system to ensure data security and integrity. Materialized views have been used to improve performance in complex queries.

The implemented solution effectively automates the download, processing, and availability of public transport data in a PostgreSQL database. The process ensures that the latest GTFS data is always available for analysis and visualization, simplifying the task for users without deep GIS or programming knowledge.

Innehållsförteckning

ABSTRACT	2
INLEDNING	5
Bakgrund	5
Problemformulering/frågeställningar	5
Syfte	5
Avgränsning	5
Teori	6
Programmering – R och RStudio	6
Funktioner	6
TryCatch	6
Databas - PostgreSQL och PostGIS	7
Transaktioner	7
Vyer	7
Common Table Expression (CTE)	8
Index	8
Schema	8
Metod och dataset	9
Schemaläggning	9
Versionshantering	9
Felhantering	9
Tabeller till databas	9
GTFS	10
Agency	11
Routes	11
Trips	11
Calendar_dates, Calendar	12
Stops	12
Stop_times	13
Shapes	14
Linjeklassificering	15
RESULTAT	16
Huvudskript	16

Funktionsskript	17
uppkoppling_db()	17
ladda_hem_gtfs()	17
skapa_tabeller()	18
versionshantering()	19
radera_gamla_versioner()	21
ladda_upp_till_databas()	21
linjeklassificering()	21
skapa_vyer_hallplats()	21
skapa_vyer_linjer()	24
skapa_vyer_historisk_hallplats/linjer()	27
Slutsatser och diskussion	29
KÄLLFÖRTECKNING	31
Källor	31
Datakällor	31

INLEDNING

Bakgrund

Avdelningen Regional utveckling inom Region Dalarna arbetar för att främja utvecklingen inom olika områden såsom näringsliv, kompetensförsörjning och samhällsbyggnad. Under Regional Utveckling finns avdelningen Samhällsanalys vars uppgift är att ta fram analyser inom dessa områden så att beslutsfattare kan fatta välgrundade beslut.

Regionen har satt upp en PostgreSQL-databas med tilläggen postGIS och pgRouting och intentionen är att fylla den med geometrisk data som ofta används i analyserna. En typ av data som ofta används är kollektivtrafikdata från Trafikförvaltningen, ett dataset som hämtas via API enligt specifikationen GTFS (General Transit Feed Specification) och som uppdateras med jämna mellanrum. Utöver avdelningen Samhällsanalys finns det även andra inom organisationen som har uttryckt behov av att enkelt kunna visualisera kollektivtrafikdata, framför allt Kollektivtrafikförvaltningen (KTF).

Problemformulering/frågeställningar

Hur kan GTFS-data i en databas tillgängliggöras för olika intressenter inom organisationen så att de enkelt kan visualiseras i ett Geografiskt informationssystem(GIS)?

Finns det ett behov av versionshantering av GTFS-data i databasen och i så fall hur kan detta implementeras?

Syfte

Syftet med arbetet är att implementera en lösning för automatisk nedladdning och tillgängliggörande av GTFS-data i en databas och att dokumentera denna. Vidare är syftet att ta fram en handledning för hur de inom regionen kan använda lösningen för att få fram den data de önskar.

Avgränsning

Den arbetsyta som skriptet var tänkt att ligga på och köras automatiskt hade blockeringar på vissa portar som behövdes. Ingen lösning hann tas fram inom projektets tidsram varpå denna del inte kommer att implementeras. Instruktioner för hur det skall implementeras när rätt portar har öppnats kommer att skickas med till Region Dalarna så att de själva kan göra det.

Teori

Programmering – R och RStudio

R är ett kraftfullt programmeringsspråk för analys och manipulering av data, **RStudio** är den integrerade utvecklingsmiljön för R som erbjuder olika verktyg för att underlätta utvecklingsarbetet. En styrka med R är möjligheten att installera olika tilläggspaket för att utöka funktionaliteten, nedan följer en beskrivning av de paket som använts i detta arbete (The R-project u.å.).

pacman - ett paket som förenklar hantering av R-paket genom att ladda, installera och uppdatera flera paket samtidigt, samt kontrollera om paket redan är installerade.

dplyr - används för att filtrera, arrangera och sammanfatta data.

httr – ett paket som underlättar interaktion med webb-API för nedladdning av data.

keyring – lagrar känslig information såsom API-nycklar och lösenord i systemets nyckelring vilket gör den osynlig i koden.

sf – hanterar och analyserar spatial data i R och bygger på Simple Features-standarden för kompatibilitet med andra GIS-verktyg.

RPostgres – ett gränssnitt mot PostgreSQL-databaser för att ansluta och utföra SQL-frågor.

glue - används för att skapa strängar genom att interpolera variabler och uttryck direkt i texten, vilket gör det enkelt att bygga dynamiska meddelanden och kodsträngar.

En **dataframe** i R är en datastruktur som används för att lagra data i tabellform med rader och kolumner. Strukturen med rader och kolumner liknar den som används i kalkylblad och databaser vilket gör den väldigt användbar för interaktion med databaser.

Funktioner

En funktion inom programmering är ett slags underprogram till det program du skriver. Dess två huvudsakliga syften är att dela upp ett större problem (program) i mindre mer lätthanterliga och förståeliga delar samt att inte behöva upprepa kod som ska köras flera gånger. En funktion kan ta emot information i form av parametrar och sedan använda denna information i den kod som tillhör funktionen. Det går även att returnera information från funktionen så att den kod som anropat funktionen kan använda denna (Sweigart 2015).

I detta arbete används funktioner för att dela upp processen i flera steg där varje funktion utför ett steg.

TryCatch

Genom att sätta ett kodavsnitt i ett try-block kan fel som uppstår när koden körs fångas upp och hanteras på ett sätt som inte avbryter körningen i ett catch-block. I detta arbete kommer tryCatch att användas tillsammans med transaktioner för att säkerställa att om någonting blir fel kommer databasen att återställas till hur den såg ut innan transaktionen inleddes.

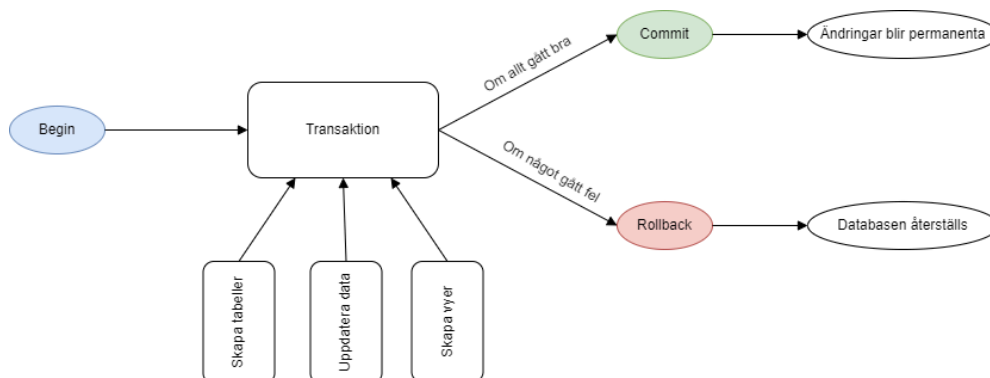
Databas - PostgreSQL och PostGIS

PostgreSQL är en avancerad objektrelationell databashanterare som har stöd för komplexa datatyper och strukturer. Den har blivit känd för att vara väldigt stabil och flexibel och erbjuder även transaktionssäkerhet och avancerade SQL-funktioner. SQL står för Structured Query Language och är det språk som används för att skicka kommandon till en databas. Den är byggd på öppen källkod och erbjuder därmed stora möjligheter för olika tillägg varav **Postgis** är ett sådant tillägg (PostgreSQL u.å.). PostGIS tillför stöd för geometriska data och rumsliga analyser och det går att på databasnivå göra beräkningar av exempelvis avstånd och area (PostGIS u.å.).

Nedan förklaras en del termer och tekniker som har använts i arbetet, samtlig information kommer från dokumentationen till PostgreSQL (PostgreSQL – Dokumentation u.å.)

Transaktioner

Transaktioner är ett fundamentalt koncept inom databashantering och ger möjligheten att bunta ihop ett flertal steg till en "allt eller inget"-transaktion. Då det är ett flertal tabeller som varje dag skall fyllas med data och sedan användas till att skapa vyer till detta data så är det viktigt att antingen sker allt eller inget.



FIGUR 1 - FLÖDET I EN DATABASTRANSAKTION

I tilläggspaketet RPostgres i R finns stöd för att implementera transaktioner med dbBegin, dbCommit och dbRollback.

Vyer

I PostgreSQL finns stöd för vyer och materialiserade vyer. Även om de bygger på samma princip finns det några skillnader i hur och när de bör användas. Användarna inom organisationen vill enkelt kunna få fram sammanställd information om busshållplatser och linjer utan att behöva skriva egen kod. De används ofta för att förenkla komplexa frågor eller för att begränsa åtkomsten till känsliga data.

Vy

En vy är en fördefinierad SQL-fråga som genererar en virtuell tabell med resultatet av frågan. Då vyerna genereras dynamiskt varje gång frågan anropas blir resultatet en aktuell bild över

data i databasen. Detta kan dock leda till prestandaproblem om frågan bygger på komplexa underliggande frågor.

Materialiserad vy

Den materialiserade vyn lagrar resultatet av frågan fysiskt i en tabell vilket är en fördel ur prestandasynpunkt, speciellt när det kommer till väldigt komplexa underliggande frågor. En nackdel är att resultatet speglar det data som fanns i databasen när vyn skapades eller uppdaterades och inte när vyn anropas. Då resultatet även lagras fysiskt i databasen kan det även bli en fråga om lagringsutrymme.

Common Table Expression (CTE)

En Common Table Expression (CTE) kan ses som en temporär tabell som används i en SQL-fråga. Genom att använda CTE:er går det att bygga komplexa frågor där resultatet av varje CTE ligger till grund för nästa CTE.

Index

Ett index är ett vanligt sätt att förbättra prestandan i en databas när det kommer till att söka och jämföra data i olika tabeller på samma sätt som ett index i en bok. Indexet lagras separat från den tabell där det är satt och innehåller pekare till det data som finns i tabellen. Även om det förbättrar prestandan när det kommer till att söka och jämföra data så kommer det också att försämra prestandan när det kommer till insättning och uppdatering av data, därför är det viktigt att vara selektiv när det kommer till hur många och vilka index som skall användas. Det finns olika typer av index, i detta arbete används B-träd för numeriska och textvärden samt GiST för geometriska data.

Schema

Ett schema är som en namngiven mapp i en databas och är ett sätt att organisera tabeller och bl.a. vyer i databasen. Det finns flertalet anledningar till att använda flera scheman i en databas istället för att lagra alla tabeller i samma. En av dessa anledningar är att olika tabeller kan ha samma namn fast i olika scheman, det går också att ställa in åtkomst eller begränsningar specifikt för vissa användare. I detta arbete används scheman för att organisera det aktuella och historiska data i separata scheman för att öka tydligheten i vad som är vad.

Metod och dataset

Under arbetets gång har samtal förts med de olika intressenterna inom organisationen vilket ligger till grund för designbesluten som tagits.

Normalt sett körs trafiken i tre olika tidtabeller under ett år; sommar (jun-aug), höst (aug-dec) och vår (dec-jun). Vid speciella tillfällen kan en extra tidtabell sättas in under året. Förutom dessa tidtabellbyten kan även mindre ändringar förekomma.

Schemaläggning

För att se till att databasen alltid är aktuell med relevant och gällande information kommer skriptet att schemaläggas att köras varje dag. För att undvika att databasen uppdateras under arbetstid sätts schemat till att köras kl. 07.15.

Versionshantering

Det finns ett behov av att kunna jämföra data mellan framför allt olika tidtabeller under årets gång men även att kunna jämföra exempelvis sommartidtabellen mellan olika år. Endast tidtabellbyten kommer att generera en ny version i databasen. Mindre ändringar som görs under en tidtabells giltighet kommer inte att versionshanteras utan istället skriva över data i tabellerna.

Felhantering

Då skriptet kommer att köras automatiskt varje natt är det viktigt att eventuella fel som uppstår hanteras på ett sätt som gör att databasens integritet inte äventyras. För att hantera eventuella fel som uppstår kommer en kombination av två tekniker att implementeras, transaktioner och felhantering med tryCatch.

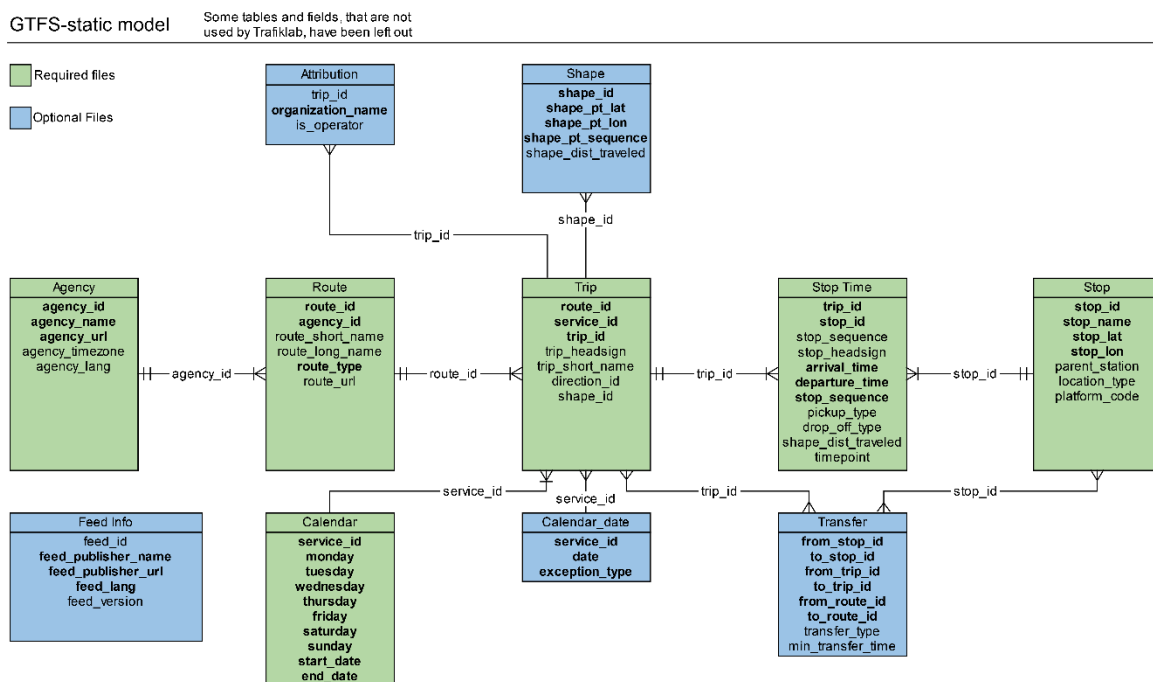
Tabeller till databas

Datasetet som laddas hem innehåller 11 st. textfiler, följande filer används inte och kommer därför inte att tas med.

- Attributions
- Feed_info
- Transfers

I genomgången av filerna nedan används begreppen primärnyckel och främmande nycklar. En **primärnyckel** är ett fält i en tabell som kan användas för att identifiera en specifik rad, det innebär att värdena i det fältet är unika. Med **främmande nycklar** menas ett fält i en tabell som är primärnyckel i en annan tabell. Genom dessa främmande nycklar är det då lätt att koppla ihop information i de olika tabellerna.

Datasetet består av elva stycken textfiler som representerar lika många tabeller varav två av dessa innehåller geometriska data (stops och shapes). I slutet av kapitlet finns en sammanställning av filerna och hur de kommer att lagras i databasen.



Relationen kan även beskrivas enligt följande:

En **route** beskriver en sträcka utmed vilken exempelvis en buss färdas, t.ex. linje 1 körs mellan Dala Airport - Borlänge centrum - Bullermyren/Övre Tjärna. En **route** körs av en **agency** och kan köras en eller flera gånger/dag. Varje **trip**(avgång) på den här ruten definieras i **trips**. Varje **trip** har ett **service_id** som i **calendar_dates** talar om vilka datum just den avgången körs.

10

Under en **trip** görs flera stopp, dessa definieras i **stop_times**. Stop_times är ett slags mellantabell som kopplar samman informationen om varje hållplats (**stop**) med de **trips** som stannar där. Förutom detta kopplas även annan information såsom ankomst- och avgångstid. Varje avgång (trip) har också ett service_id som i **calender_dates** talar om vilka datum avgången körs. En geometrisk linje från **shapes** som beskriver vägen kopplas också samman i denna tabell (Trafiklab 2022).

Exempel: Linje 1 mellan Dala Airport - Borlänge centrum - Bullermyren/Övre Tjärna med avgång kl. 10.00 stannar vid 28 hållplatser. Det ger 1 **trip** med 28 hållplatser (**stops**) sammankopplade i tabellen **stop_times** 28 gånger.

Agency

Primärnyckel: agency_id

Beskrivning

Innehåller förutom Dalatrafik 6 observationer (Länstrafiken Örebro, VL m.fl.) som ej används i övriga data.

Slutsats

Kan överföras som den är till databasen.

Routes

Primärnyckel: route_id

Främmande nyckel: agency_id

Beskrivning

Route_type: Använder "Extended Route Types" och värdena 100 (Tåg), 700 (Buss) och 1501 ("Communal Taxi Service"/Flexområde). Används dock ej konsekvent i datasetet varpå en egen klassificering baserad på route_short_name kommer att införas, se Linjeklassificering.

Slutsats

Överförs som den är till databasen efter att data har konverterats till rätt datatyp.

Trips

Primärnyckel: trip_id

Främmande nycklar: route_id, service_id, shape_id

Beskrivning

Innehåller förutom nycklarna trip_headsign som inte används och direction_id (1 eller 0) som anger i vilken riktning på rutten avgången körs.

Övrigt

Ingen övrig relevant information.

Slutsats

Överförs som den är till databasen efter att direction_id konverterats till heltal.

Calendar_dates, Calendar

Primärnyckel: service_id, date

Beskrivning

Calendar är tom, istället används calendar_dates där varje datum ett service_id körs anges.

Övrigt

Calendar tas med även om den inte används utifall den i senare dataset finns med.

Slutsats

Båda tabellerna överförs som de är till databasen efter att date konverterats till datatypen date och exception_type till heltal.

Stops

Primärnyckel: stop_id

Beskrivning

stop_lat, stop_lon: Koordinaten för hållplatsen/hållplatsläget.

location_type: 1 för **Hållplats** och 0 för **Hållplatsläge**.

parent_station: Ett stop_id som refererar till en hållplats (location_type 1) till vilken hållplatsläget hör.

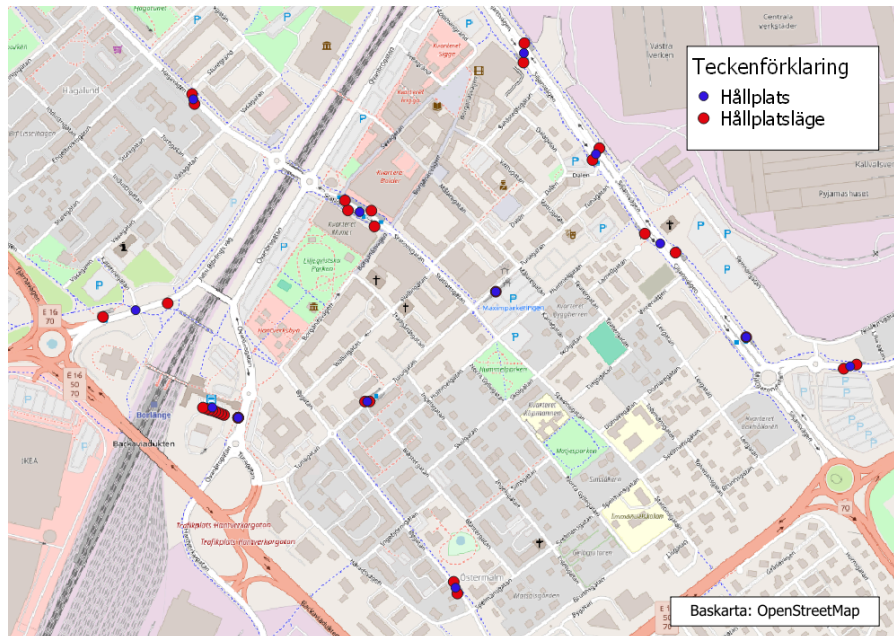
stop_id	stop_name	stop_lat	stop_lon	location_type	parent_station	platform_code
9021020585268000	ABB Ludvika rv 50	60.143960	15.179141	1		
9022020585268001	ABB Ludvika rv 50	60.143896	15.178962	0	9021020585268000	A
9022020585268002	ABB Ludvika rv 50	60.144023	15.179302	0	9021020585268000	B

FIGUR 3 - HÅLLPLATSEN ABB LUDVIKA RV 50 MED TVÅ HÅLLPLATSLÄGEN

Övrigt

Stop_lat och stop_lon konverteras om till POINT innan överföring till databasen.

Hållplatsläge är den fysiska hållplatsen där en buss stannar. **Hållplats** är centrumkoordinaten för de tillhörande hållplatslägena.



FIGUR 4 – VISUELL BESKRIVNING AV HÅLLPLATSER OCH HÅLLPLATSLÄGEN

Slutsats

Överförs som den är till databasen efter konvertering av koordinaten och location_type konverterats till heltal.

Stop_times

Primärnyckel: trip_id, stop_sequence

Främmande nycklar: trip_id, stop_id

Beskrivning

Kombinationen av trip_id och stop_sequence bildar primärnyckel.

Pickup_type och drop_off_type: Indikerar huruvida en resenär måste indikera för på- eller avstigning. Time_point: Visar om tiden för ankomst/avgång är exakt(1) eller ungefärlig(0).

Övrigt

Ankomst- och avgångstid kan ibland vara större än 24:00 för nattrafik. Till exempel en buss som avgår kl. 01:00 natten mellan fredag och lördag har avgången på fredagen och tiden satt till 25:00, trafikdygn är 04:00:00 till 03:59:59

Slutsats

Pickup_type_drop_off_type, timepoint och stopsequence konverteras till heltal, shape_dist_traveled till flyttal. Tidsangivelserna behålls som text för att behålla tider större än 24:00.

Shapes

Primärnyckel: shape_id, shape_pt_sequence.

Främmande nycklar: Inga

Beskrivning

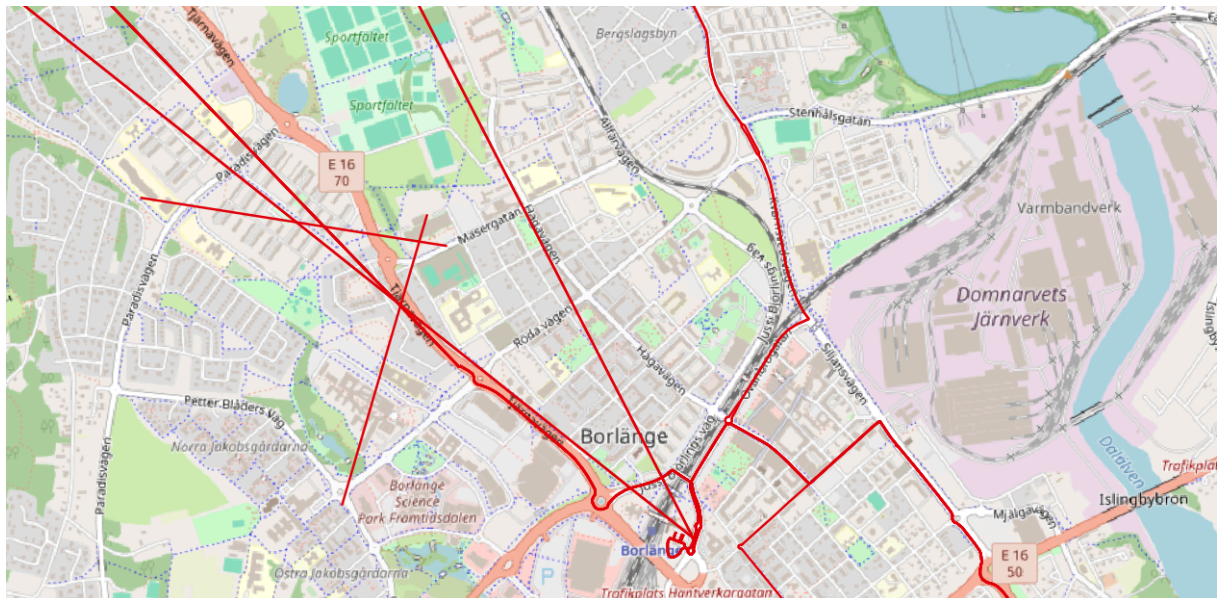
Kombinationen av shape_id och shape_pt_sequence bildar primärnyckel. Varje shape består av en samling punkter med longitud och latitud samt ett heltal som visar i vilken ordning punkterna ligger. Det finns även ett fält för hur långt avstånd det är mellan punkten och starten. Sammantaget beskriver varje shape en linje utmed vilken fordonet färdas.

shape_id	shape_pt_lat	shape_pt_lon	shape_pt_sequence	shape_dist_traveled
1205330000102897684	60.430010	15.507647	1	0
1205330000102897684	60.430047	15.507691	2	0.01
1205330000102897684	60.430041	15.507704	3	0.99
1205330000102897684	60.430040	15.507706	4	1.08
1205330000102897684	60.429724	15.508407	5	53.31

FIGUR 5 - EN SHAPE BESTÅR AV ETT ANTAL GEOMETRISKA PUNKTER I EN SEKvens

Övrigt

Det finns ca 1,6 miljoner rader i tabellen då varje "linje" består av tusentals punkter i en bestämd ordning. Det finns drygt 100 shapes som är felaktiga i det att de i vissa fall endast består av 1-2 punkter, andra följer vägen en del av sträckan men tar ett stort hopp mellan två punkter.



FIGUR 6: FELAKTIGA LINJER I SHAPES.

Slutsats

För att minska antalet rader i databasen görs varje shape om till en LINESTRING innan de överförs till databasen, det ger 1394 rader istället för 1,6 miljoner. I samband med detta skapas även fält för hur många punkter linjen är skapad av samt ett fält med den största distansen mellan två punkter. På så vis kan felaktiga linjer enkelt identifieras och filtreras bort. Den nya tabellen får namnet shapes_line.

Linjeklassificering

Primärnyckel: route_short_name

Främmande nycklar: Inga

Beskrivning

Det finns ett behov av att kunna filtrera linjerna/rutterna beroende på vad för typ av trafik det är. Då route_type i routes inte används konsekvent samt att behov finns av ytterligare klassificeringar skapas en tabell som kopplar route_short_name (linjenumret) med vad för typ av trafik det är.

Slutsats

Varje route_short_name läggs till i en tabell med dess tillhörande klassificering enligt följande:

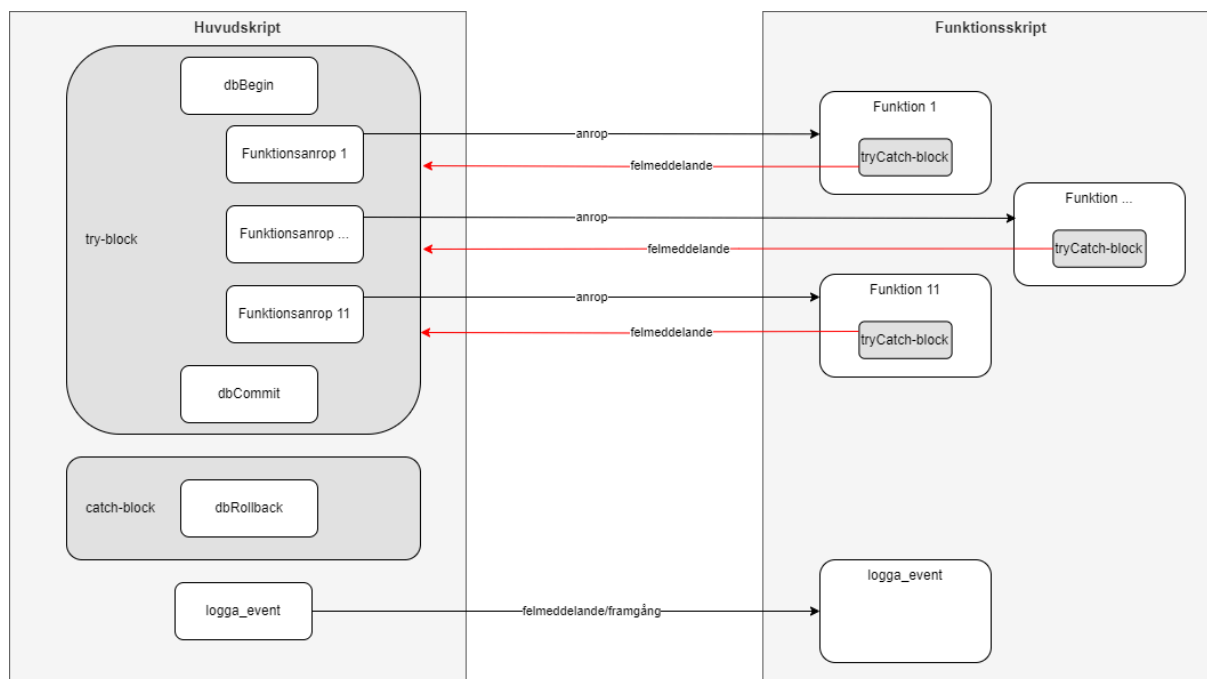
Stadstrafik	1 – 99 och 151-154
Landsbygdstrafik	100 – 400 och 39
Flextrafik	430 - 800
Stängd skoltrafik	900 - 1000
Stråktrafik	101-199
Utomlänstrafik	42, 46, 241, 360, 500

RESULTAT

Arbetet resulterade i två st. filer med R-kod, härafter kallade huvudskript och funktionsskript (Bilaga 1). Huvudskriptets funktion är att anropa funktionerna i funktionsskriptet, hantera databastransaktionen och att logga eventuella fel som uppstått eller om processen har fullföljts framgångsrikt. Funktionsskriptet består av 12 stycken funktioner som utför varje steg i processen, funktionerna ansvarar även för att avbryta körningen om något gått fel och att rapportera vad och var felet uppstod till huvudskriptet.

Huvudskript

Då skriptet körs automatiskt varje morgon stod det klart tidigt att en robust felhantering behövde implementeras. Hela processen består av flera steg som vart och ett bygger på tidigare steg och tabellerna har även relationer med varandra vilket innebär att det är av stor vikt att antingen sker allt eller så skall ingenting ske. Genom att kombinera transaktioner med tryCatch ser skriptet till att om ett fel uppstår någonstans i processen så skall körningen avbrytas och databasen återställas till hur den såg ut innan skriptet började köras.



FIGUR 7 - INTERAKTIONEN MELLAN DE TVÅ SKRIPTEN GÄLLANDE FELHANTERING

Funktionsskript

Funktionsskriptet är där alla funktionerna är implementerade och det är de som gör själva arbetet. Processen kan beskrivas enligt följande:

1. Läs in paket som används i koden
2. Skapa en uppkoppling till databasen
3. Ladda hem GTFS-data via API
4. Skapa tabeller i databasen om de inte redan finns
5. Versionshantering – finns det en ny tidtabell?
6. Radera gamla versioner
7. Ladda upp GTFS-data till databas
8. Skapa eller uppdatera tabellen med linjeklassificeringar
9. Skapa vyer för hållplatser och linjer
10. Skapa vyer för hållplatser och linjer för historiska data/äldre tidtabeller

Nedan följer en beskrivning av vad de olika funktionerna i funktionsskriptet gör. Samtliga funktioner använder tryCatch för att fånga upp eventuella fel och skicka ett beskrivande meddelande till huvudskriptet varvid detta inte nämns i beskrivningen av funktionerna.

uppkoppling_db()

Denna funktion etablerar en säker anslutning till en PostgreSQL-databas genom att använda RPostgres-paketet tillsammans med keyring-paketet för autentisering. Funktionen tar följande parametrar:

- **keyring_service_name**: Namnet på nyckelringstjänsten som innehåller autentiseringsuppgifter (standardvärde är "rd_geodata").
- **db_host**: Värnamnet eller IP-adressen till databasservern (standardvärde är "WFALMITVS526.ltdalarna.se").
- **db_port**: Portnumret för databasservern (standardvärde är 5432).
- **db_name**: Namnet på databasen (standardvärde är "praktik").
- **db_options**: Ytterligare anslutningsalternativ, exempelvis sökväg (standardvärde är "-c search_path=public").

Vid körning försöker funktionen ansluta till databasen med hjälp av de specificerade parametrarna. Användarnamn och lösenord hämtas från systemets nyckelring, vilket säkerställer att känsliga uppgifter inte exponeras i koden.

Funktionen returnerar ett anslutningsobjekt (**con**) om anslutningen lyckas.

ladda_hem_gtfs()

Denna funktion laddar ner, packar upp och läser in GTFS-data från Trafiklab. GTFS-data levereras som en uppsättning kommaseparerade textfiler där var och en representeras av en tabell i databasen. Efter att ha konverterat de olika kolumnerna till rätt datatyp returneras data i en lista med dataframes (**gtfs_data**) där varje element i listan motsvarar en databastabell. Funktionen tar inga parametrar.

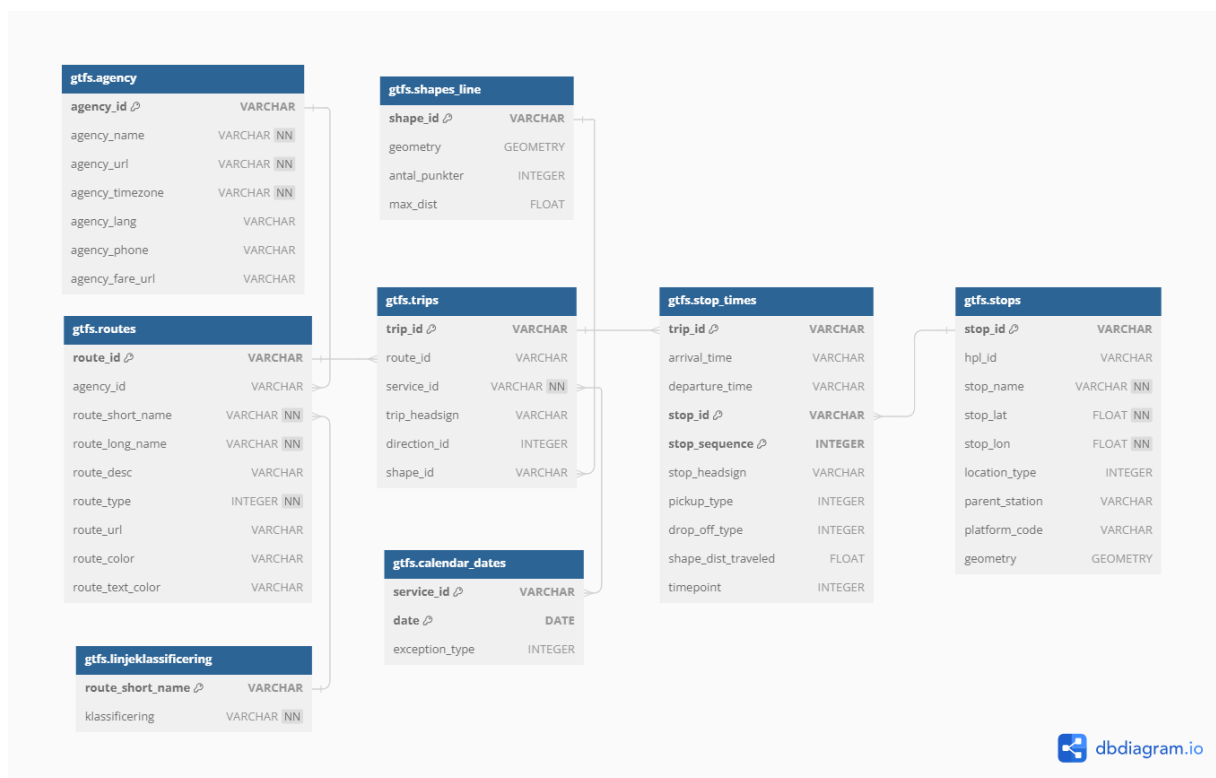
skapa_tabeller()

I funktionen skapas nödvändiga scheman och tabeller i databasen för att lagra aktuell GTFS-data samt historisk data. Funktionen tar följande parameter:

con: En aktiv anslutning till databasen.

Funktionen skapar scheman för aktuell och historisk data, **gtfs** och **gtfs_historisk**. Sedan skapas i schemat **gtfs** med korrekta datatyper för alla fält om de inte redan finns:

- **agency:** Innehåller information om operatörer.
- **routes:** Innehåller information om rutter, inklusive index på **route_short_name**.
- **calendar_dates:** Innehåller datum då avgångar körs.
- **shapes_line:** Innehåller geometriska former för rutter, inklusive ett GIST-index på **geometry**.
- **trips:** Innehåller information om avgångar, inklusive index på **shape_id**, **route_id**, och **service_id**.
- **stops:** Innehåller information om hållplatser, inklusive deras geometri och ett GIST-index på **geometry**.
- **stop_times:** Innehåller tidtabeller för varje hållplatsstopp, inklusive index på **trip_id** och **stop_id**.
- **linjeklassificering:** Innehåller klassificering av rutter.



FIGUR 8 - ER-DIAGRAM FÖR SCHEMAT GTFS

Liknande tabeller skapas även i schemat **gtfs_historisk** för att hantera historisk data med tillägg av en kolumn **version** för versionshantering.

Slutligen skapas en tabell **versions** i det historiska schemat för att lagra information om de olika historiska versionerna.



FIGUR 9 - ER-DIAGRAM FÖR SCHEMAT GTFS_HISTORISK MED TABELL OCH FÄLT FÖR VERSIONSHANTERING

Funktionen returnerar inget värde men utför SQL-kommandon för att skapa och indexera tabellerna i databasen. Då samtliga SQL-kommandon använder **IF NOT EXISTS** sker detta i praktiken bara första gången funktionen körs.

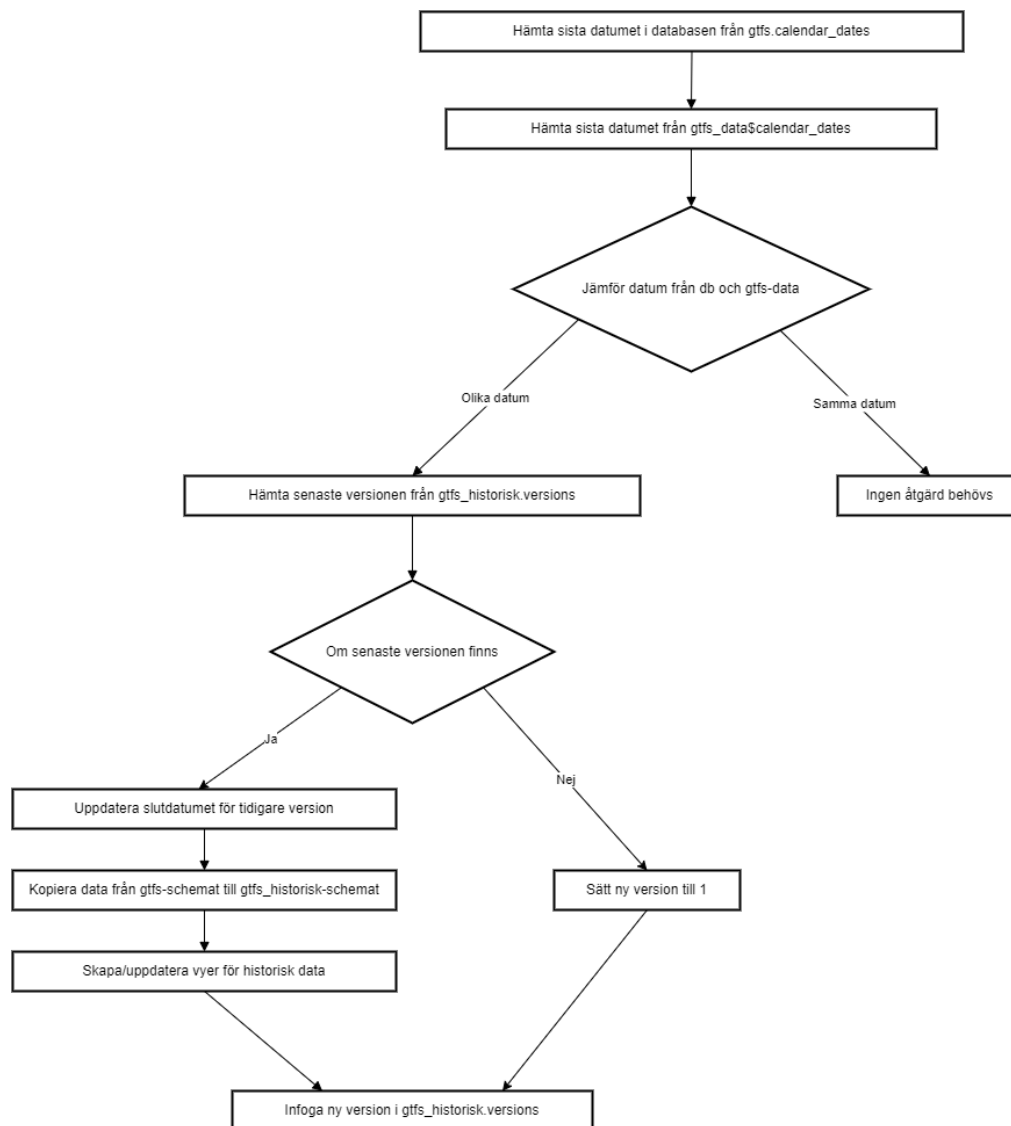
versionshantering()

Denna funktion hanterar versionshantering av GTFS-data i databasen. Om den aktuella GTFS-data har ett senare sistadatum än den som finns i databasen, skapar funktionen en ny version och kopierar data från det aktuella schemat (gtfs) till det historiska schemat (gtfs_historisk). Funktionen tar följande parametrar:

con: En aktiv anslutning till databasen.

gtfs_data: Listan med dataframes som representerar den inlästa GTFS-data, kommer från funktionen `ladda_hem_gtfs()`.

Algoritmen kan beskrivas med följande flödesschema.



FIGUR 10 - FLÖDESSCHEMA ÖVER VERSIONSHANTERING

Funktionen börjar med att hämta och jämföra de sista datumen från både det nedladdade gtfs-data samt det som är sparat i databasen. Om dessa skiljer sig åt innebär det att det antingen är ett tidtabellsskifte eller att databasen är tom, om de inte skiljer sig åt innebär det att det enbart är en ny uppsättning data inom samma tidtabell och därför inte skall versionshanteras. I nästa steg kontrolleras ifall det redan finns en version i versionstabellen, om det inte gör det skall den första versionen skapas med startdatum satt till det första datumet i gtfs-datat. Om det finns en tidigare version i databasen innebär det att det data som finns i schemat gtfs skall sparas ner i det historiska schemat och sedan skall vyerna för det historiska datat antingen skapas eller uppdateras (se funktion skapa_vyer_historisk_hallplats/linjer).

radera_gamla_versioner()

Denna funktion tar bort data från gamla versioner av GTFS-data som är äldre än vad som anges i år i parametern **antal_ar** från det historiska schemat i PostgreSQL-databasen. Funktionen tar följande parametrar:

- **con**: En aktiv anslutning till databasen.
- **antal_ar**: Antalet år efter vilka äldre versioner skall raderas från databasen.

Funktionen returnerar ingenting men jämför `end_date` i tabellen `versions` med dagens datum minus antalet år som anges i **antal_ar** och raderar sedan data från alla tabeller för de versioner som är äldre än det. På så sätt sparas endast relevant historisk data i databasen vilket förbättrar prestandan och minskar lagringskostnaden.

ladda_upp_till_databas()

Denna funktion laddar upp GTFS-data från en lista med dataframes till en PostgreSQL-databas. Funktionens huvudsakliga syfte är att tömma de befintliga tabellerna och sedan fylla dem med ny data, inklusive att hantera spatial data för hållplatser och rutter. Funktionen tar följande parameter:

- **con**: En aktiv anslutning till databasen.
- **gtfs_data**: Listan med dataframes som representerar den inlästa GTFS-datan, kommer från funktionen `ladda_hem_gtfs()`.

Då versionshanteringen redan har skett är det riskfritt att tömma tabellerna med aktuell data innan det nya datat laddas upp för att på så vis se till att databasen alltid är uppdaterad med den senaste informationen. I tabellen `stops` används `stop_lat` och `stop_lon` för att skapa en punkt (point) innan den förs över till databasen. Tabellen `shapes` som egentligen består av en massa punkter i en sekvens görs om så att det blir linjer (linestring) istället, samtidigt beräknas antalet punkter samt det största avståndet mellan två punkter och sparas i två nya fält för att kunna identifiera eventuella felaktiga linjer.

linjeklassificering()

Funktionens uppgift är att skapa och fylla tabellen **linjeklassificering** med klassificeringar av rutter baserad på den lista som erhållits från KTF och attributet **route_short_name** i **routes**. Funktionen tar följande parametrar:

- **con**: En aktiv anslutning till databasen.

Funktionen returnerar ingenting men möjliggör visning av vilken typ av trafik som trafikerar en hållplats eller rutt.

skapa_vyer_hallplats()

Funktionen skapar två stycken materialiserade vyer för att sammanställa information om avgångar per hållplatsläge och hållplats och vilka linjer som trafikerar dem. Funktionen tar följande parameter:

- **con**: En aktiv anslutning till databasen.

vy_hallplatslage_avgangar

De första materialiserade vyn som skapas är den för hållplatslägen. Det fanns behov av att kunna se antalet avgångar, vilka linjer (rutter) som trafikerar hållplatsläget samt vilken klassificering dessa linjer har. Valet föll på att skapa en materialiserad vy då SQL-frågan som sammanställer informationen från alla tabeller blev komplex resurskrävande.

Den materialiserade vyn sammanställer informationen från ett antal CTE:er för att visa genomsnittliga antal avgångar/vardag samt antal avgångar på lördagar respektive söndagar. Vilka linjer som trafikerar hållplatserna och deras klassificering, i vyn kallad linjetyp, tas också med. En kort beskrivning av CTE:erna följer nedan.

SENASTE_VERSION

Hämtar startdatum för den senaste versionen så att enbart trafik fr.o.m. detta datum tas med.

DAGLIGA_AVGANGAR

Beräknar antalet avgångar per dag (datum) för varje hållplatsläge genom att koppla samman information från tabellerna stop_times, trips och calendar_dates. Endast datum fr.o.m. startdatum från senaste_version tas med i beräkningen.

AVGANGAR_VECKA

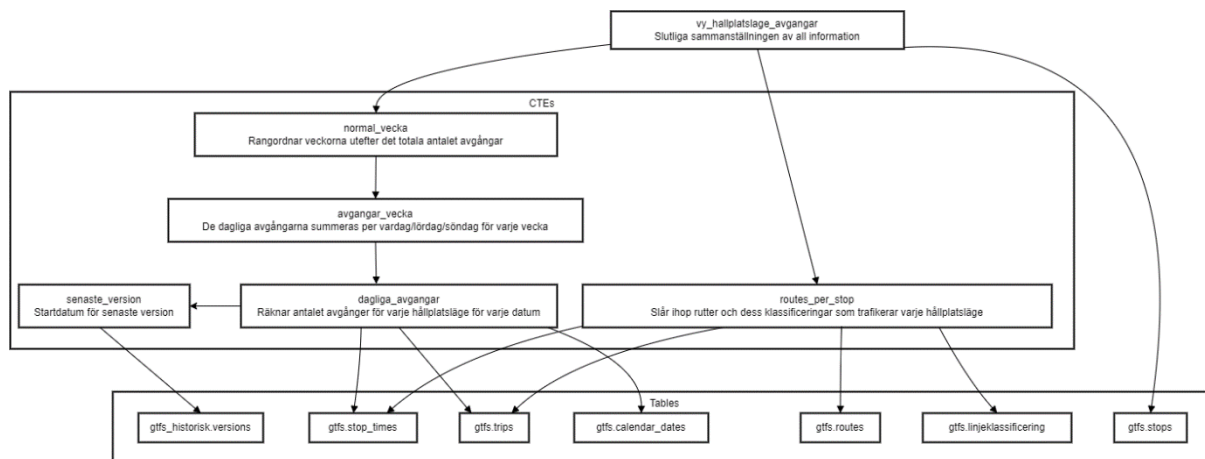
Summerar antalet avgångar för vardag, lördag och söndag grupperat på vecka från dagliga_avgangar.

NORMAL_VECKA

Rangordnar veckorna från avgangar_vecka utefter den som har flest avgångar för hela veckan. Att välja den veckan med flest avgångar som "normal vecka" gjordes utifrån antagandet att det som oftast ändrar antalet avgångar en vecka är om det infaller en röd dag under en vardag och antalet avgångar på söndagar alltid är färre än på en vardag.

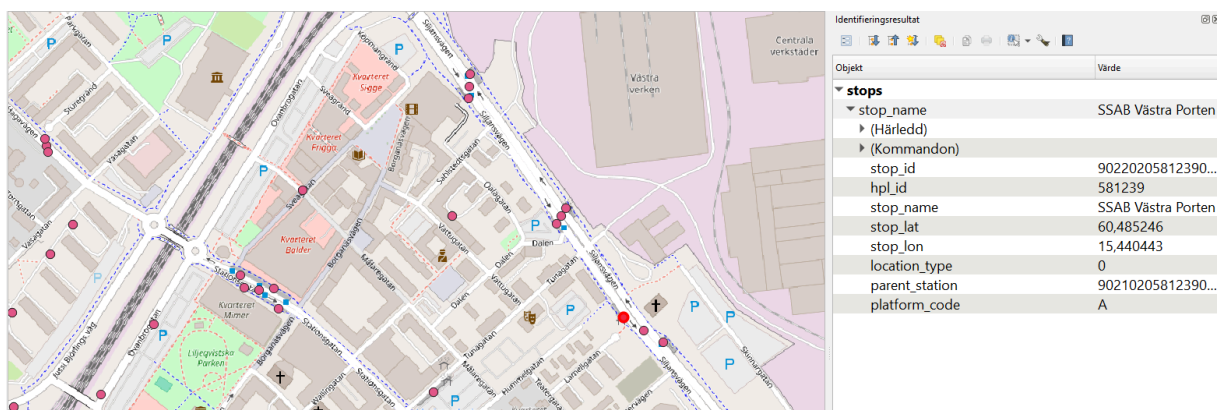
ROUTES_PER_STOP

Sammanställer information från tabellerna stop_times, trips, routes och linjeklassificeringar för att skapa kommaseparerade textsträngar med alla linjer och deras linjetyper, för varje hållplatsläge.

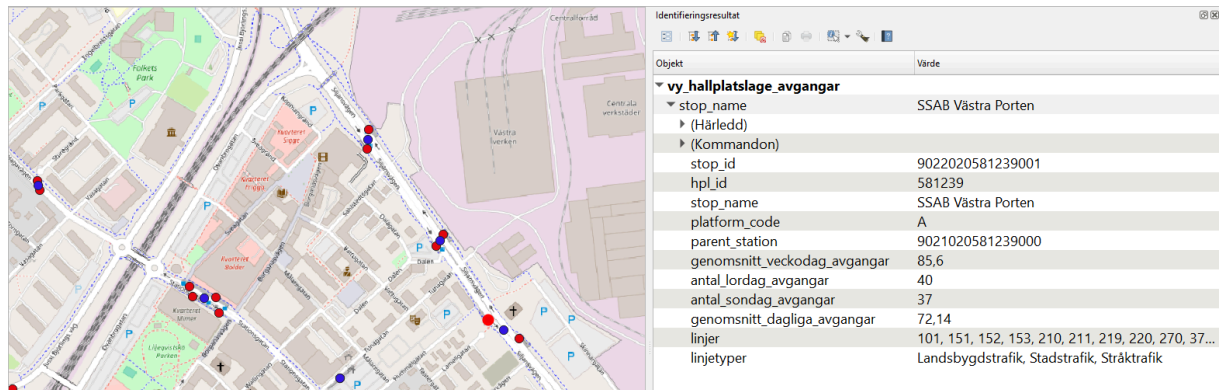


FIGUR 11 - FLÖDESDIAGRAM ÖVER VY_HALLPLATSLAGE_AVGANGAR

De två figurerna nedan visar skillnaden mellan den information som visas i tabellen stops jämfört med informationen som visas i vy_hallplatslage_avgangar.



FIGUR 12 - INFORMATION OM ETT HÅLLPLATSLÄGE FRÅN TABELLEN STOPS



FIGUR 13 - INFORMATION OM ETT HÅLLPLATSLÄGE FRÅN VY_HALLPLATSLAGE_AVGANGAR

vy_hallplats_avgangar

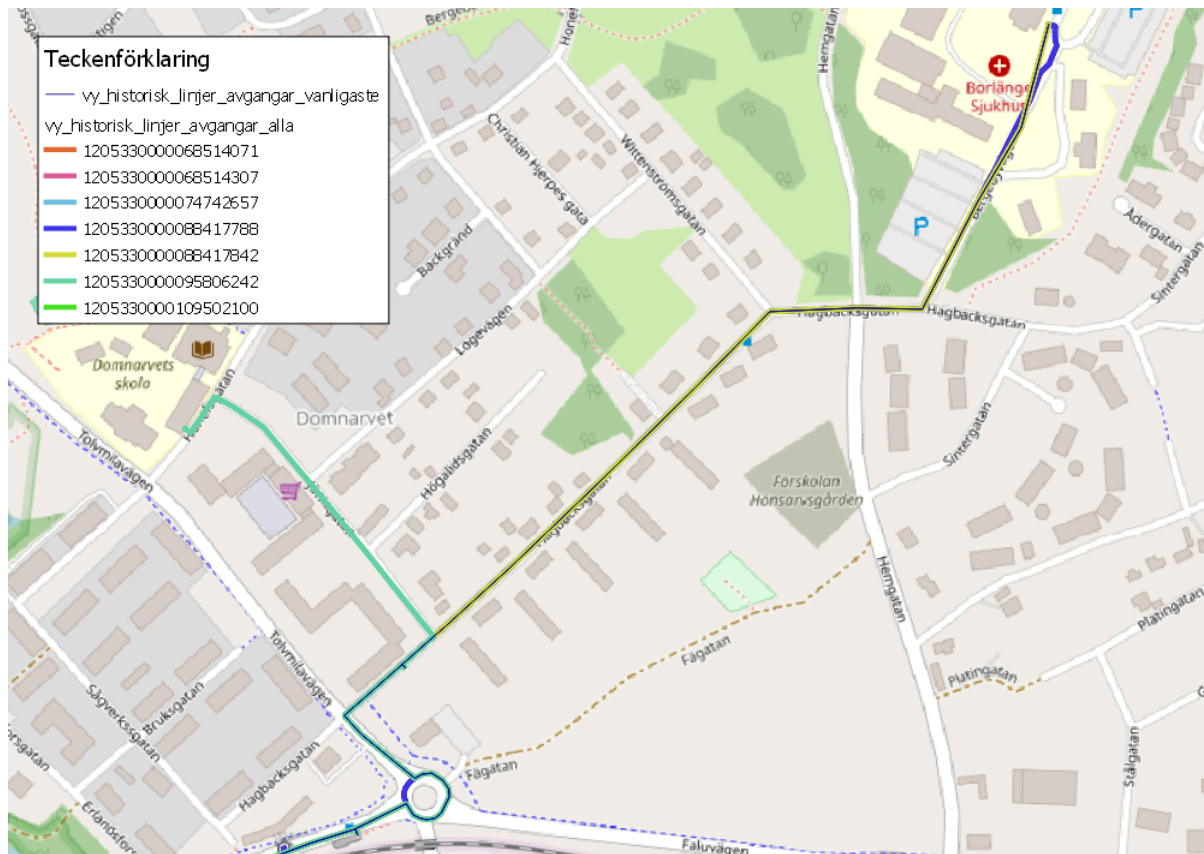
Den andra materialiserade vyn som skapas i funktionen är den som sammanställer informationen på hållplatsnivå och bygger uteslutande på vy_hallplatslage_avgangar. I många analyser används hållplats istället för hållplatsläge och därför summeras informationen om antal avgångar och linjer för de lägen som hör till varje hållplats.

skapa_vyer_linjer()

Denna funktion skapar två stycken materialiserade vyer för sammanställa information om rutter och de geometriska linjer som representerar färdvägen för en rutt. Funktionen tar följande parameter:

- **con:** En aktiv anslutning till databasen.

En rutt (route) kan ta flera olika färdvägar beroende på avgång (trips), det kan vara olika ändhållplatser eller att de tar olika vägar. Det innebär att för exempelvis rutt nr 2 finns det 7 stycken olika geometriska linjer (shapes_line) vilka representerar de olika färdvägarna (se figur 11). Det finns ett behov i verksamheten av att kunna se alla olika färdvägar för en rutt men också den vanligaste färdvägen varpå två vyer har implementerats.



FIGUR 14 - DE SJU OLIKA FÄRDVÄGARNÄ FÖR RUTT NR 2 I VY_LINJER_ALLA OCH DEN FÄRDVÄGEN SOM HAR FLEST AVGÄNGAR I VY_LINJER_VANLIGASTE

vy_linjer_alla

Den första materialiserade vyn sammanställer information om antal avgångar per geometrisk linje på liknande sätt som vyerna för hållplatser. Den bygger på ett antal CTE:er, vilka beskrivs kort nedan.

SENASTE_VERSION

Hämtar startdatum för den senaste versionen så att enbart trafik fr.o.m. detta datum tas med.

DAGLIGA_AVGANGAR

Beräknar antalet avgångar per dag (datum) för varje geometrisk linje genom att koppla samman information från tabellerna trips och calendar_dates. Endast datum fr.o.m. startdatum från senaste_version tas med i beräkningen.

AVGANGAR_VECKA

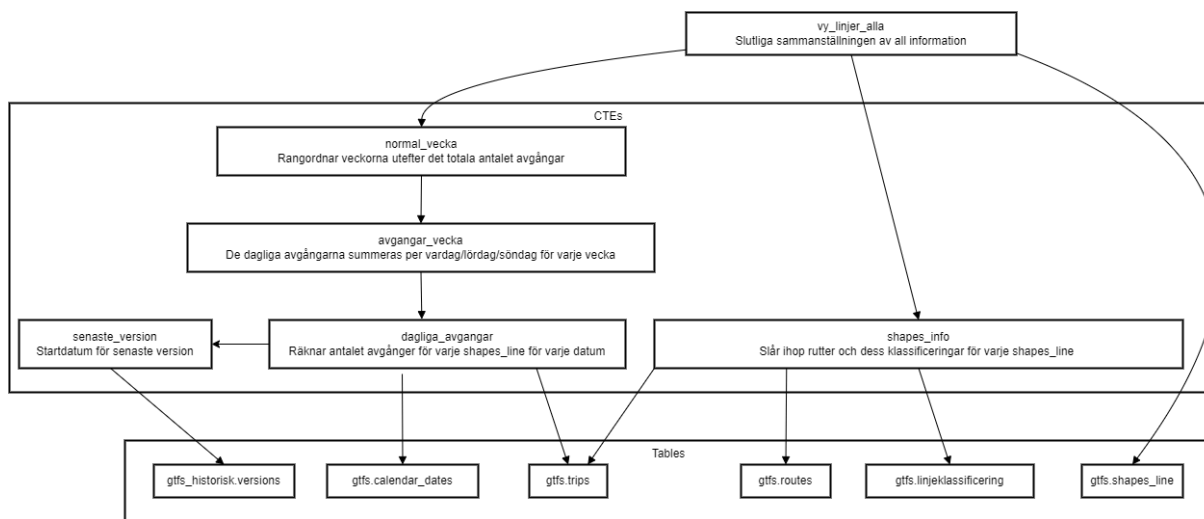
Summerar antalet avgångar för vardag, lördag och söndag grupperat på vecka från dagliga_avgangar.

NORMAL_VECKA

Rangordnar veckorna från avgangar_vecka utefter den som har flest avgångar för hela veckan.

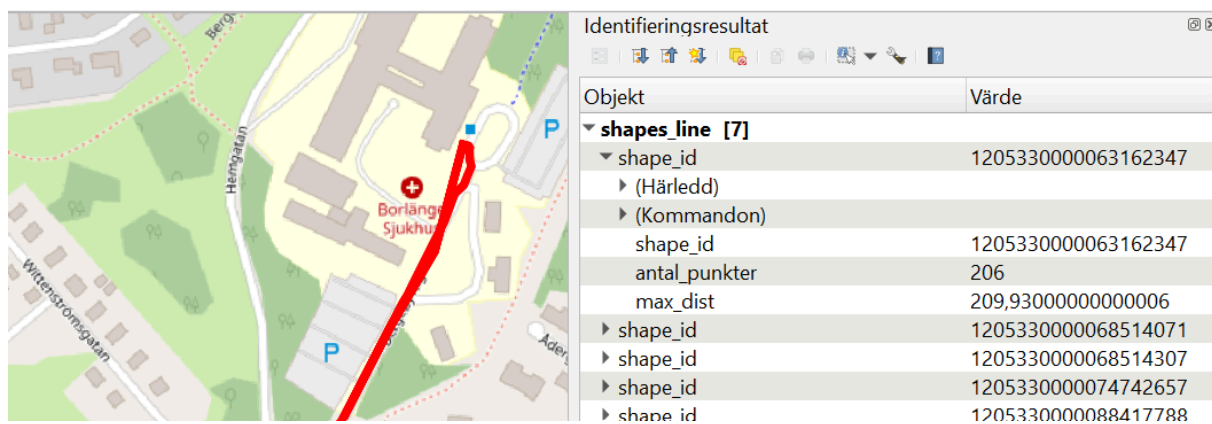
SHAPES_INFO

Kopplar samman information om vilken rutt som kör varje färdväg och vilken linjetyp det är med själva geometriska linjen. Här beräknas även antalet avgångar som är kopplade till varje geometrisk linje.

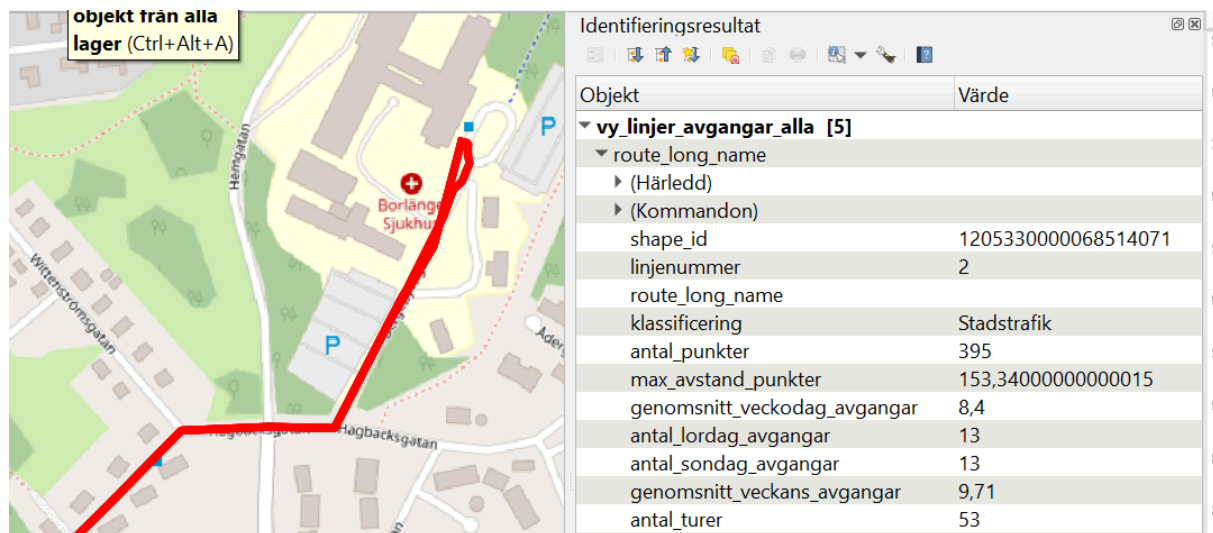


FIGUR 15 - FLÖDESDIAGRAM ÖVER VY_LINJER_VANLIGASTE

De två följande figurerna visar skillnaden i information som visas för tabellen shapes_line och vy_linjer_vanligaste.



FIGUR 16 - INFORMATION OM EN GEOMETRISK LINJE I TABELLEN SHAPES_LINE.



FIGUR 17 - INFORMATION OM EN GEOMETRISK LINJE I VY_LINJER_ALLA.

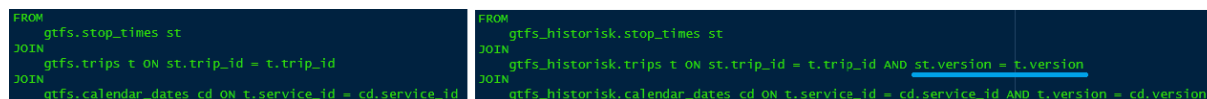
vy_linjer_vanligaste

Den andra materialiserade vyn i funktionen är den som visar den vanligaste färdvägen för en rutt, d.v.s. för varje rutt visas enbart en geometrisk linje. Den bygger uteslutande på den information som sammanställts i `vy_linjer_alla` och beskrivs därmed inte i närmare detalj. Den geometriska linje som har flest antal avgångar som använder sig av den färdvägen är den som väljs ut som den vanligaste för varje rutt.

skapa_vyer_historisk_hallplats/linjer()

Funktionsskriptet innehåller två funktioner till vars uppgift är att skapa motsvarande vyer för historiska data. De är också de enda funktionerna som inte anropas från huvudskriptet utan från funktionen `versionshantering`. Anledningen till att de inte anropas från huvudskriptet är för att den enda gången de behöver uppdateras är när det har skapats en ny historisk version och data har lagts till i det historiska schemat.

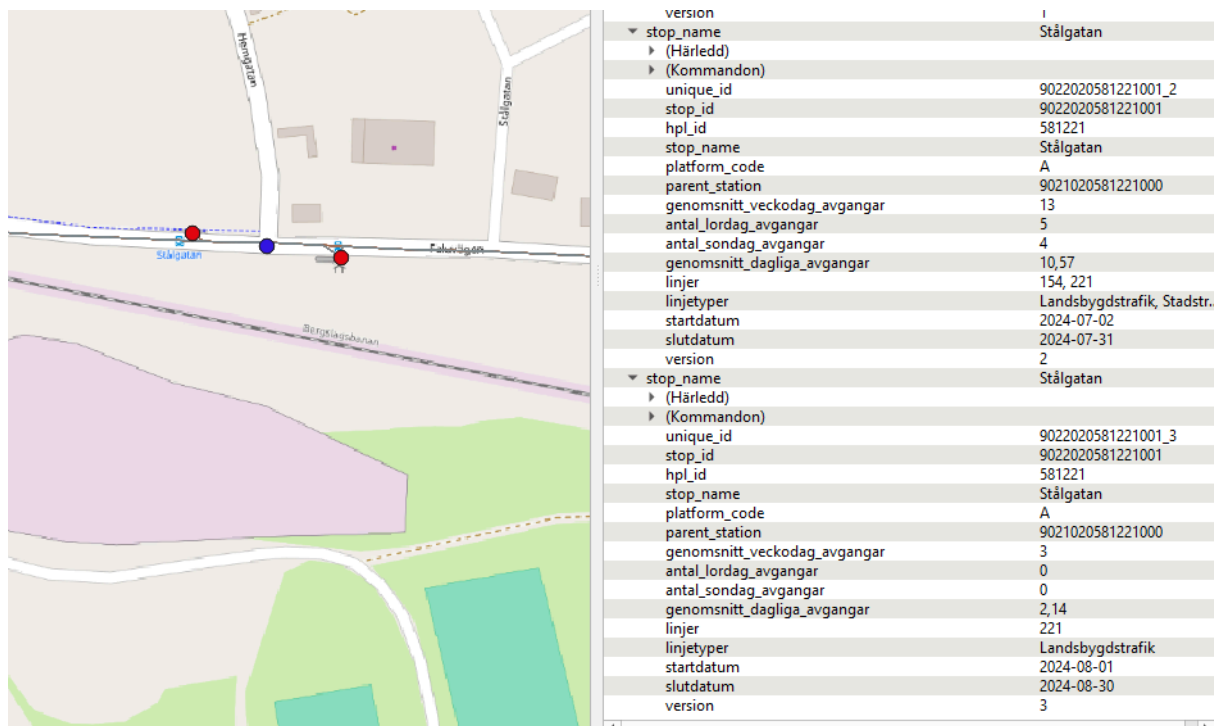
Skapandet av vyer är identiskt med vyerna för det aktuella gtfs-datat med den enda skillnaden att i varje sammankoppling av information från flera tabeller så tas även versionsnumret med. Genom att ta med versionsnumret i varje koppling garanteras det att, vid exempelvis summering av antal avgångar på en hållplats, enbart information för just den tidtabellen/versionen tas med och inte för alla tidtabeller.



FIGUR 18 - SKILLNADEN I EN KOPPLING FÖR AKTUELL DATA (VÄNSTER) OCH HISTORISK DATA (HÖGER).

När informationen i de historiska vyerna sammanställs så tas start- och slutdatum för de olika historiska versionerna med. Detta gör att det blir enklare att filtrera ut en specifik version utan

att behöva titta i versionstabellen för vilka datum versionen gäller för. I figuren nedan syns skillnaden mellan version 2 och 3 i fråga om vilka linjer och deras klassificeringar(linjetyper) som trafikerar hållplatsläget vilket också ger ett färre antal avgångar.



FIGUR 19 - INFORMATION SOM VISAS OM ETT HÅLLPLATSLÄGE I VY_HISTORISK_HALLPLATSLAGE_AVGANGAR

Slutsatser och diskussion

Den implementerade lösningen har visat sig effektiv för att automatiskt ladda ner, bearbeta och tillgängliggöra kollektivtrafikdata i en PostgreSQL-databas. Genom att använda R och de olika tilläggsprogrammen för att interagera med databasen har en process skapats som säkerställer att den senaste GTFS-datan alltid finns tillgänglig för analys och visualisering. Genom att sammanställa informationen från de olika tabellerna i materialiserade vyer så kan även användare utan djupgående GIS- eller programmeringskunskaper snabbt och enkelt visualisera och analysera kollektivtrafikdata. De flesta moderna GIS erbjuder någon form av databaskoppling och genom att skapa vyer är det lika enkelt att hämta in dessa som ett lager som det är att hämta in en tabell.

Kollektivtrafikförvaltningen uttryckte ett behov av att kunna jämföra data mellan olika tidtabeller och år, därför inkluderar den implementerade lösningen även en form av versionshantering där historiska data lagras i ett eget schema i databasen. Beslutet att lägga historiska data i egna tabeller i ett eget schema togs utifrån både prestanda och användbarhet. Då aktuell data är det som används klart mest är det viktigt att ur ett prestandaperspektiv hålla de tabellerna så små som möjligt. Genom att lägga det i ett eget schema med suffixet `_historisk` blir det också tydligt i ett GIS vilken uppsättning data det är. Även om representanter för KTF endast uttryckte behov av att kunna se historiska data 1 år tillbaka i tiden sattes antalet år till 3 innan äldre data raderas. Mängden data i de historiska tabellerna kommer inte att på 3 år bli så pass stor att det kommer att påverka prestandan nämnvärt men samtidigt erbjuda lite flexibilitet utifrån de skulle behöva äldre data. Antalet år är satt som en parameter till funktionen som raderar data vilket ger dem möjlighet att enkelt kunna ändra denna siffra ifall de skulle få prestandaproblem. Denna funktionalitet har testats med en kortare tidsangivelse än 3 år för att se att det fungerar som tänkt.

Vyerna som presenterar historiska data tar med startdatum och slutdatum för de olika tidtabeller som versionerna representerar så att det enkelt går att filtrera ut en äldre tidtabell. När testningen skulle genomföras för de historiska vyerna fanns det bara en historisk version i databasen. För att kunna testa att den fungerade som tänkt kopierades all data i denna version och sedan skapades två till versioner utifrån denna data. För att inte alla tre versioner skulle vara identiska raderades en del rader i databasen som kopplade avgångar till hållplatser (se figur 19, s. 28), ett liknande test genomfördes med avgångar kopplade till geometriska linjer. Utfallet blev korrekt och som förväntat.

Skriptet fungerar bra att köra som ett automatiserat och schemalagt skript, det har testats på utvecklingsdatorn. När portarna på den arbetsytan där skriptet är tänkt att ligga har öppnats bör det enkelt kunna gå att flytta över det dit och sedan schemaläggas där. Under projektets gång skedde ett tidtabellsbyte vilket gjorde att det även kunde testas i skarpt läge. Skriptet hanterade flyttningen av det gamla aktuella datat till det historiska schemat och uppdateringen av det aktuella datat på ett förväntat och korrekt sätt.

Att använda materialiserade vyer istället för vyer förbättrade prestandan oerhört mycket. När tester utfördes med vanliga vyer i ett GIS låste sig programmet i flera minuter och i vissa fall blev jag tvungen att starta om. Frågorna och CTE:erna som vyerna bygger på är komplexa som hämtar och sammankopplar data från flera olika tabeller vilket gör det rimligt ur en prestandasynpunkt att använda materialiserade vyer då de sparar ner resultatet i en tabell

när vyn uppdateras. Den enda anledningen till att använda vanliga vyer skulle vara för att få den mest aktuella informationen direkt från databasen. Eftersom databasen endast uppdateras när skriptet körs och de materialiserade vyerna uppdateras direkt efter att ny data importerats faller det argumentet. Således ger användandet av materialiserade vyer både bättre prestanda men också tillräckligt aktuell information för de behov som finns. Genom att sätta index på de kolumner i databasen som används i skapandet av vyerna har även här en förbättring i prestandan noterats. Index har även satts på de geometriska kolumnerna vilket bör förbättra prestandan i spatiala analyser och jämförelser mot andra geometriska objekt, detta har dock inte testats.

Felhanteringen med användandet av tryCatch och transaktioner säkerställer databasens integritet genom att avbryta skriptet om något skulle gå fel i processen. Ingenting sparas permanent i databasen förens alla steg har gått igenom utan problem. Om något skulle gå fel så återställs databasen till hur den såg ut innan skriptet började köras. Oavsett om skriptet körs igenom i sin helhet med framgång eller om det avbryts kommer detta att loggas i en fil tillsammans med datum och eventuella fel-meddelanden som genererats.

Praktiska tillämpningar och nytta för intressenter

En praktisk tillämpning av informationen som sammanställs i vyerna är möjligheten att filtrera på vad för typ av trafik som trafikerar en hållplats och hur många turer som stannar där. I en tillgänglighetsanalys kan då exempelvis endast hållplatser som har helgtrafik inkluderas och hållplatser som har färre än X antal avgångar/vecka exkluderas. Det finns möjlighet att identifiera de hållplatser som endast används till "flextrafik" och exkludera dessa i vidare analyser. När det kommer linjer går det på samma sätt att filtrera på typ av trafik men även här på antal avgångar, vilka linjer som blir påverkade av ett planerat vägarbete eller en tidskrävande olycka på ett vägvsnitt går snabbt att få fram. I bilaga 2 finns en användarhandledning med fler praktiska exempel på filtreringar och användande.

De som inte har så djupa kunskaper i GIS kan nu själva ta fram och visualisera data och utföra dessa analyser utan att behöva beställa detta från exempelvis avdelningen Samhällsanalys. Detta skulle då minska väntan för beställaren och arbetsbördan för analytikern.

Framtida förbättringar och utvecklingsområden

Förhoppningen är att detta arbete kan inspirera utvecklingen av fler automatiserade skript som aktualiserar och visualiserar data i en databas. Om det framkommer behov av ytterligare information om hållplatser och linjer kan fler vyer skapas eller de som finns vidareutvecklas.

En tanke som fanns med ganska tidigt för loggfunktionen är att det bör skickas ett mail till databasansvarig om det uppstår ett fel under körningen så att hen kan åtgärda eller åtminstone informera om att databasen kanske inte är aktuell.

Vidare och mer omfattande testning av framför allt de historiska vyerna när databasen populerats med fler versioner med riktig data är också något som kan ligga lite framåt i tiden.

KÄLLFÖRTECKNING

Källor

The R-project (u.å.). *What is R?*

<https://www.r-project.org/about.html> [2024-06-02]

Sweigart, A. (2015). *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. No Starch Press.

Tillgänglig på <https://automatetheboringstuff.com/> [2024-06-19]

PostgreSQL (u.å.). *About*

<https://www.postgresql.org/about/> [2024-06-02]

PostGIS (u.å.)

<https://postgis.net/> [2024-06-02]

PostgreSQL – Dokumentation (u.å.)

<https://www.postgresql.org/docs/> [2024-06-24]

Trafiklab (2022) *Static GTFS files*.

<https://www.trafiklab.se/sv/docs/using-trafiklab-data/using-gtfs-files/static-gtfs-files/> [2024-06-30]

Datakällor

GTFS Regional Static Data

API tillgängligt via:

<https://opendata.samtrafiken.se/gtfs/{operator}/{operator}.zip?key={apikey}>



Bilagor

Bilaga 1 - Kod

All kod i projektet finns tillgänglig på:
https://github.com/mikael-leonidsson/gtfs_till_postgis

Bilaga 2 – Användarhandledning

Användarhandledning – GTFS till PostGIS

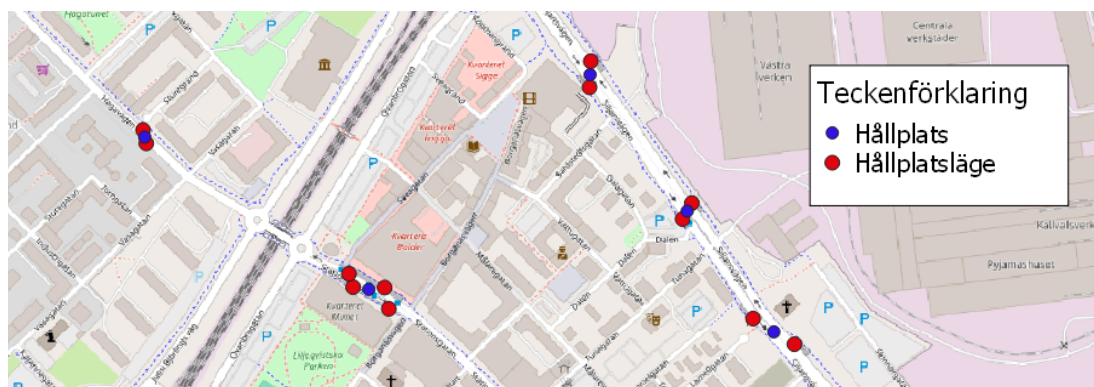
Nedan följer en genomgång av de vyer som implementerats i databasen. De fält som finns tillgängliga finns beskrivna och hur informationen har sammanställts. Det finns även exempel på olika filtreringar som kan användas. Exempelen är från QGIS och funktionen Filter, se slutet av dokumentet för hur vyer kan användas i R.

Schema gtfs

I schemat gtfs finns informationen för den gällande tidtabellen. Det innehåller två tabeller med geometrisk data, stops (hållplatser) och shapes_line (geometriska linjer). Utöver dessa två tabeller finns det även 4 st vyer där information är sammanställd för hållplatser och hållplatslägen samt linjer.

Hållplatser och hållplatslägen

Både hållplatser och hållplatslägen kommer från tabellen stops, skillnaden ligger i att ett Hållplatsläge har ett **stop_id** i fältet **parent_station**. Detta stop_id refererar till dess kopplade Hållplats.



vy_hallplatslage_avgangar

Följande fält finns i vyn:

Fält	Beskrivning
Stop_id:	Id från tabellen stops. Ser till att det finns ett unikt fält i vyn
Hpl_id:	Dalatrafikens hållplatsID. Deriverat från stop_id – tecken 8-13
Stop_name:	Hållplatsens namn
Platform_code:	Hållplatslägets kod/perong.
Parent_station:	stop_id för den Hållplats läget tillhör.
Genomsnitt_veckodag_avgangar:	Antalet avgångar Mån-Fre / 5 under en "normalvecka" *
Antal_lordag_avgangar:	Antalet avgångar Lör under en "normalvecka" *
Antal_sondag_avgangar:	Antalet avgångar Sön under en "normalvecka" *

Genomsnitt_veckans_avgangar:	Antalet avgångar Mån-Sön / 7 under en "normalvecka"
Linjer:	Kommaseparerad textsträng med vilka linjenummer som trafikerar hållplatsläget
Linjetyper:	Kommaseparerad textsträng med klassificeringen på de linjer som trafikerar hållplatsläget

* "Normalvecka" beräknas enligt följande: vyn summerar alla avgångar/datum för varje hållplatsläge. Sedan delas dessa avgångar upp veckovis. Slutligen väljs den veckan med flest avgångar som "normalvecka" enligt resonemanget att det den veckan inte har några röda dagar i sig.

Vy_hallplats_avgangar

Vyn bygger på vy_hallplats_lage och summerar all information för de hållplatslägen som tillhör hållplatsen.

Fält	Beskrivning
Stop_id:	Id från tabellen stops. Ser till att det finns ett unikt fält i vyn
Hpl_id:	Dalatrafikens hållplatsID. Deriverat från stop_id – tecken 8-13
Stop_name:	Hållplatsens namn
Genomsnitt_vekodag_avgangar:	Antalet avgångar Mån-Fre / 5 under en "normalvecka"
Antal_lordag_avgangar:	Antalet avgångar Lör under en "normalvecka"
Antal_sondag_avgangar:	Antalet avgångar Sön under en "normalvecka"
Genomsnitt_veckans_avgangar:	Antalet avgångar Mån-Sön / 7 under en "normalvecka"
Linjer:	Kommaseparerad textsträng med vilka linjenummer som trafikerar hållplatsen
Linjetyper:	Kommaseparerad textsträng med klassificeringen på de linjer som trafikerar hållplatsen

Exempel på filtreringar

Hitta hållplatser som trafikeras på lördagar **och** söndagar.

Specifikt filteruttryck för datakälla

```
"antal_lordag_avgangar" > 0 AND "antal_sondag_avgangar" > 0
```

Vill man istället ha hållplatser med trafik på lördagar **eller** söndagar.

Specifikt filteruttryck för datakälla

```
"antal_lordag_avgangar" > 0 OR "antal_sondag_avgangar" > 0
```

Hållplatser med enbart 'Flextrafik' – dvs ingen annan trafik stannar vid hållplatsen (observera att linjetyper innehåller en textsträng, därför måste sökvärden stå innanför ")

Specifikt filteruttryck för datakälla

```
"linjetyper" = 'Flextrafik'
```

Hållplatser som **inte** enbart är 'Flextrafik' – hållplatser som har ex.vis Flextrafik och Stadstrafik kommer med i urvalet.

Specifikt filteruttryck för datakälla

```
"linjetyper" != 'Flextrafik'
```

För att filtrera hållplatser på linjetyper där t.ex. 'Stadstrafik' är en av de trafiktyper som stannar vid hållplatsen behöver man använda sig av **ILIKE** och "**wildcards**" (%). Går precis som med alla andra villkor att använda de logiska operatorerna AND/OR. I exemplet nedan filtrerar vi ut alla hållplatser som har antingen 'Stråktrafik' **eller** 'Landsbygdstrafik'. OBS! Det finns även LIKE, skillnaden är att ILIKE ignorerar stora och små bokstäver.

Specifikt filteruttryck för datakälla

```
"linjetyper" ILIKE '%Stråktrafik%' OR  
"linjetyper" ILIKE '%Landsbygdstrafik%'
```

Motsvarigheten till **inte lika med** (!=) när man använder ILIKE och wildcards är **NOT ILIKE**. Så för att ta fram hållplatser som inte har någon 'Stadstrafik' blir filtreringen:

Specifikt filteruttryck för datakälla

```
"linjetyper" NOT ILIKE '%Stadstrafik%'
```

När det kommer till att filtrera på linjenummer blir det lite krångligare då dessa sparas som text. Det innebär att om vi bara använder t.ex. LIKE '%52%' så kommer vi även få med linje 152, 252, 520 o.s.v. Det finns två sätt att lösa det på, att sätta upp ett antal LIKE som matchar 52 på flera sätt eller använda en regexp.

Exempel med LIKE, observera att efter kommatecken är det alltid ett blanksteg:

Specifikt filteruttryck för datakälla

```
linjer LIKE '52,%'      -- Börjar med 52
OR linjer LIKE '%, 52'  -- Slutar med 52
OR linjer LIKE '%, 52,%' -- Innehåller 52 någonstans i mitten
OR linjer = '52'        -- Enbart 52
```

Ett annat sätt är att använda regexp vilket är ett sätt att matcha teckenkombinationer i en sträng.

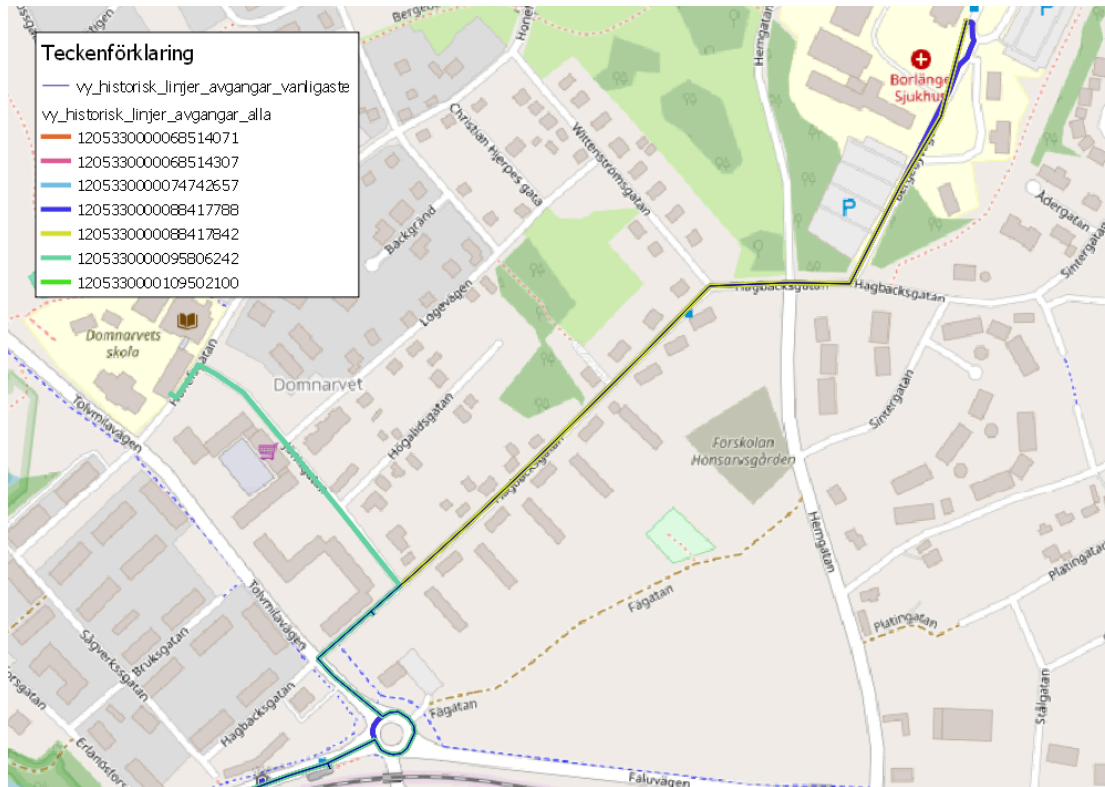
Specifikt filteruttryck för datakälla

```
linjer ~* ' (^|, )52(, |$) '
```

Förklaring: Sök efter 52, framför ska det vara början av strängen (^) eller (|) ett komma följt av blanksteg (,). Efter ska det vara ett komma följt av ett blanksteg (,) eller (|) slutet av strängen (\$).

Linjer

Det finns två vyer för linjer, en för alla geometriska linjer och en för den linje med flest turer för varje rutt.



De 7 st olika linjerna som används av rutt/linjenummer 2 och den av dessa som har valts ut som den vanligaste (vy_historisk_linjer_avgangar_varligaste OBS! Ignorera att det är den historiska vyn, den funkar på samma sätt)

vy_linjer_avgangar_alla

Följande fält finns i vyn:

Fält	Beskrivning
shape_id:	Id från tabellen stops. Ser till att det finns ett unikt fält i vyn
linjenummer:	Vilken rutt/linjenummer använder sig av linjen
Route_long_name:	Används enbart för Tåg
klassificering:	Klassificeringen på linjenumret som använder linjen
Antal_punkter:	Antalet punkter linjen består av – kan användas för att hitta "felaktiga linjer"
Max_avstand_punkter:	Det största avståndet mellan två punkter på linjen – kan användas för att hitta "felaktiga linjer"
Genomsnitt_veckodag_avgangar:	Antalet avgångar Mån-Fre / 5 under en "normalvecka" *
Antal_lordag_avgangar:	Antalet avgångar Lör under en "normalvecka" *
Antal_sondag_avgangar:	Antalet avgångar Sön under en "normalvecka" *
Genomsnitt_veckans_avgangar:	Antalet avgångar Mån-Sön / 7 under en "normalvecka" *
Antal_turer:	Hur många turer (trips) använder sig av linjen

* "Normalvecka" beräknas enligt följande: vyn summerar alla avgångar/datum för varje hållplatsläge. Sedan delas dessa avgångar upp veckovis. Slutligen väljs den veckan med flest avgångar som "normalvecka" enligt resonemanget att det den veckan inte har några röda dagar i sig.

Fälten antal_punkter och max_avstand_punkter kan användas för att identifiera linjer som är "felaktiga" – dvs de består antingen av ett fåtal punkter eller har ett stort glapp mellan två punkter. I exemplet nedan visas också på möjligheten att gruppera filter genom att använda parenteser. Välj alla linjer som INTE är Flextrafik OCH (antingen består av färre än 10 punkter eller har ett avstånd mellan 2 punkter som är över 2 km).

Specifikt filteruttryck för datakälla

```
"klassificering" != 'Flextrafik'
AND ("antal_punkter" < 10
OR "max_avstand_punkter" > 2000)
```

vy_linjer_avgangar_vanligaste

Följande fält finns i vyn:

Fält	Beskrivning
shape_id:	Id från tabellen stops. Ser till att det finns ett unikt fält i vyn
linjenummer:	Vilken rutt/linjenummer använder sig av linjen
Route_long_name:	Används enbart för Tåg

klassificering:	Klassificeringen på linjenumret som använder linjen
Antal_punkter:	Antalet punkter linjen består av – kan användas för att hitta "felaktiga linjer"
Max_avstand_punkter:	Det största avståndet mellan två punkter på linjen – kan användas för att hitta "felaktiga linjer"
Genomsnitt_veckodag_avgangar:	Antalet avgångar Mån-Fre / 5 under en "normalvecka" *
Antal_lordag_avgangar:	Antalet avgångar Lör under en "normalvecka" *
Antal_sondag_avgangar:	Antalet avgångar Sön under en "normalvecka" *
Genomsnitt_veckans_avgangar:	Antalet avgångar Mån-Sön / 7 under en "normalvecka" *

I denna vy har den linje med störst **Antal_turer** i vy_linjer_avgangar_alla valts ut som den linje som representerar varje rutt/linjenummer. I övrigt är de identiska.

Filtreringen i dessa vyer fungerar på samma sätt som för hållplatserna med undantaget att man inte behöver använda **ILIKE**, då alla fält enbart innehåller ett värde används **=/!=** för att jämföra.


Specifikt filteruttryck för datakälla

```
"linjenummer" = '2'
```

Schema gtfs_historisk

Så fort systemet har upptäckt en ny tidtabell i det nedladdade datasetet kommer all data från tabellerna i schemat gtfs att kopieras över till motsvarande tabeller i det historiska schemat och tilldelas ett versionsnummer. I det schemat finns även en tabell som håller koll på start- och slutdatum för varje version. För tillfället finns bara en tidigare version av gtfs i databasen, men detta kommer att fyllas på så fort det byts tidtabell.

De historiska vyerna funkar på exakt samma sätt som vyerna för aktuell data i schemat gtfs med en väsentlig skillnad. När man klickar på exempelvis en hållplats så kommer alla historiska versioner att synas, dvs varje punkt är egentligen 3 punkter i det här fallet. Då systemet testades skapades 3 kopior av den enda historiska versionen i databasen – sedan togs en del avgångar bort i version 3. Se bilden nedan, Information om version 1 och 3 visas och i version 3 har alla avgångar från hållplatsen för linje 154 tagits bort.



Identifieringsresultat

Objekt	Värde
vy_historisk_hallplats_avgangar [3]	
stop_name	Stålgatan
(Härledd)	
(Kommandon)	
unique_id	9021020581221000_1
stop_id	9021020581221000
hpl_id	581221
stop_name	Stålgatan
start_date	2024-02-27
end_date	2024-06-16
version	1
genomsnitt_veckodag_avgangar	29,8
antal_lordag_avgangar	9
antal_sondag_avgangar	7
genomsnitt_dagliga_avgangar	23,57
linjer	154, 210, 221
linjetyper	Landsbygdstrafik, Stadstrafik
stop_name	Stålgatan
stop_name	Stålgatan
(Härledd)	
(Kommandon)	
unique_id	9021020581221000_3
stop_id	9021020581221000
hpl_id	581221
stop_name	Stålgatan
start_date	2024-08-01
end_date	2024-08-30
version	3
genomsnitt_veckodag_avgangar	6
antal_lordag_avgangar	0
antal_sondag_avgangar	0
genomsnitt_dagliga_avgangar	4,28
linjer	210, 221
linjetyper	Landsbygdstrafik

För att enklare kunna filtrera ut rätt version har start- och slutdatum lagts till i alla historiska vyer. För att exempelvis hitta alla hållplatser som trafikerades av Stadstrafik i våras kan följande filter användas, välj ett datum inom den period du är intresserad av och sätt att startdatum ska vara mindre än det och slutdatum större än det datumet.

Specifikt filteruttryck för datakälla

```
linjetyper ILIKE '%Stadstrafik%'
AND startdatum < '2024-04-01'
AND slutdatum > '2024-04-01'
```

Vyer i R

Att hämta en vy via R och SQL är precis som att hämta en vanlig tabell via SQL. Antingen hämtas hela vyn eller så kan man filtrera på samma sätt som i QGIS

```
library(mapview)
hallplats_avgangar <- st_read(con, query = "SELECT * FROM gtfs.vy_hallplats_avgangar;")
hallplats_stadstrafik <- st_read(con, query = "SELECT * FROM gtfs.vy_hallplats_avgangar
                                             WHERE linjetyper ILIKE '%Stadstrafik%';")

mapview(hallplats_avgangar)
mapview(hallplats_stadstrafik)
```


Resultat för hållplatser med Stadstrafik

