Sebastian Brauer

GIS 5577

Dr. David Haynes

April 16th, 2024

**Final Project Paper**

**Introduction:**

Earlier this semester the Met Council got in touch with me and my advisor Ying Song about producing web maps that highlight the ‘flow’ of people throughout the Twin Cities using the 2019 Travel Behavior Inventory. The goal of this visualization was to use it as an exploratory data analysis tool to better understand overall trip trends in the metro area. The data was provided through the MN Geospatial Commons data portal from the Metropolitan Council and is composed of eight separate tables of which five were leveraged in the database. Lastly, the 2010 census block groups were also downloaded from the Commons. The current architecture of this dataset is one that lends itself well to being implemented as a PostgreSQL database because of the large number of tables and keys that connect these tables. With the ability to connect the tables, new analyses can be run to better understand the various characteristics of people and households and the types of trips they complete within the Twin Cities greater metro area. Vitally, a PostgreSQL environment would serve well to created origin/destination matrices from the trip data while adding data from other tables.

**Background:**

In 2019, the Metropolitan Council, the metropolitan planning organization or MPO for the greater seven county twin cities area, released the results of their Travel Behavior Inventory. The TBI was a travel data survey that was conducted between one to seven days, depending on the collection method, and totaled 351,177 trips for 16,152 different participants across 7,837 households. Historically, travel survey data of this nature is hard to collect at large scales such as this due to the complex nature of the data being collected, the expensive collecting infrastructure needed, and the challenges that can arise when participants are collecting their own data. Due to privacy concerns, the 6,830,105 GPS trip location points have been aggregated to the block group level.

Overall, the database is comprised of five aspatial tables and one spatial dataset which contains the block group boundaries from the 2010 census for the seven-county metropolitan area. The five tables are Day, Household, Person, Trip, and Vehicle and the spatial dataset is a polygon layer projected in WGS84 with an ESPG: 4326 called tc\_bg\_2010census. Every table has a unique primary key, some of which appear in other tables as foreign keys to connect the tables together in the database. The Day table has person\_day\_id, Household has hh\_id, Person has person\_id, Trip has trip\_id, Vehicle has vehicle\_id, and tc\_bg\_2010census has geoid10.

Trip serves as the ‘keystone’ table as it holds the foreign keys of three other tables Household, Person, and tc\_bg\_2010census. It has a primary key of trip\_id and holds information about legs of linked trips, mode, purpose, distance, duration, and speed in addition to other columns. The foreign keys present are hh\_id, person\_id, and o\_bg/d\_bg which correspond to the origin and destination block group geoid10 for that certain trip. It is important to note is that each trip also has a linked\_trip\_id and this value is important to determine the entire trip and these values were used when creating the origin destination matrices for the flow maps. Linked trips correspond to entire trips that may have multiple legs, such as a walk to the bus stop before taking the bus, then walking to another bus stop and transferring before arriving to work. This entire trip is composed of multiple legs which each have their own trip\_id but the same linked\_trip\_id.

Household is the most ‘general’ table and can easily be joined to other tables at is has the least rows of any tables. Its primary key is hh\_id and it also contains information about the number of people in the house, their occupation (job/student), household income, type of residence, whether the people rent, how long they’ve lived there. The foreign keys present are person\_id and home\_bg correspond to the block group geoid10 for that certain home. Person contains more demographic information about the specific people that live in the households. The primary key is person\_id and contains information about the person age, ethinicty, student status, and gender. The tc\_bg\_census2010 spatial dataset with its primary key being geoid10. It also contains the geometry of the Block Groups. Day and Vehicle play a more minor role in the database. Day serves as a way to check which days a certain person\_id was partaking in the data. Since the person\_id can be tied directly to the hh\_id, the hh\_id foreign key is also present in the data. Lastly, Vehicle has a primary key of vehicle\_id and contains data regarding vehicle age, fuel type, and ownership type and serves as additional metrics that can be attached to certain trips.

**Methods:**

As mentioned in the introduction, all of these tables came from the MN Geospatial Commons and originally contained many more fields, but the fields listed in this report, the final ER diagram, and the database. Those fields were removed as they were not needed for the analysis. For the TBI tables, the tables were created using CREATE TABLE statements and copied over from csv files that were locally stored. These CREATE TABLE statements can be found in the linked sql file. Lastly, the tc\_bg\_census2010 shapefile was added to the PostgreSQL database using the shptosgsql-gui. It must be noted that the original shapefile from the MN Geospatial Commons was reprojected from UTM Zone 15N (ESPG: 26915) to WGS84 (ESPG: 4326).

**Results:**

The goal of this database was to create origin destination matrices of trips between census block groups not simply for trips overall but also while being able to query for certain values of columns from other tables. For example, querying trips made by people that are only part of a low-income household, setting income\_broad to ‘<$25K’ from Household. What made this analysis more complicated was the presence of multi-legged trips within Trips. Because the flow of the trip is the goal a way to extract the o\_bg of the first leg of the trip and the d\_bg of the last leg of the trip was needed. This was achieved in example queries 3a,b,c from the example queries file linked by using a series of Common Table Expressions. The first block creates a table with each unique linked\_trip\_id with the value of the first and last leg in FirstLastLegs, next the o\_bg of the first leg and the d\_bg of the last leg are extracted in FirstLegOrigins and LastLegOrigins. Lastly, AggregatedTrips takes the d\_bg from LastLegOrigins and joins it to the FirstLegOrigins table and then uses a GROUP BY on those three fields to get a table containing a list of unique linked trips with the o\_bg from the first leg and the d\_bg from the last leg.

With this table of unique linked\_trip table created called AggregatedTrips, the trips and proper foreign key (either hh\_id or person\_id depending on query) were joined back to AggregatedTrips using SELECT DISTINCT linked\_trip ids. That foreign key is then used to join the proper table. If the desired field was already in the trips table, then that fields was joined instead of a foreign key. One calculation required a couple additional calculations prior to running the main query and that was to get the trip mode. Since each trip or leg of a linked trip has its own mode, there needs to be a way to group that mode when aggregating by linked\_trip\_id. The methodology used in this was to use whichever trip mode was used for a majority of the distance of that trip. Query 3c highlights two more additional CTE tables that were created prior to running the query. First TotalDistances uses a GROUPBY from trips on the linked\_trip\_id and mode group while summing the distance. Second, MaxTotalDistance then grouped the TotalDistances table by linked\_trip\_id and got the MAX total\_distance value. That max distance value was then used as the join key back to join to TotalDistance and the AggregatedTrips. The result is a table with linked\_trip\_id, o\_bg, d\_bg, mode, and max\_distance travelled by that mode. This table was then used to create origin destination matrices for trips while querying by mode.

**Discussion & Conclusion:**

This project was not without its challenges. The complicated nature of the data structure made joining tables and aggregating tables the correct way very challenging. The nature of the linked trips within Trips provided an additional challenge. The most challenging aspect of the work was working with all of the joins and understanding the role of select distinct when joining. When using ‘Joins&Relates’ in ArcGIS Pro, you have the option of choosing rules on how you join tables when joining from a table with more features to one with less like FIRST() or SUM(), such as joining trip data to linked trip data. This one – to – max or one – to – sum joins would be very helpful in this process. Simply using JOIN or even INNER JOIN had the same number of rows as the original table with more rows, trips in this example. This idea of having rules when joining as present in ArcGIS Pro was not as available in the PostgreSQL environment. Needing to use a SELECT DISTINCT first to match the number of rows between both tables proved helpful. This is especially the case when we know that all of the hh\_id values are the same for each linked\_trip\_id. Thanks to this project I now have a much better understanding of how joins work. The other pieces of information that was important for this work was type casting between bigint and varchar. When adding a shapefile to the PostgreSQL database, the varchar data type is used for integers instead of bigint.

In conclusion, this database sought out to solve the problem presented at the start of this report. The database was used to create 19 different origin destination matrices with different attributes about household location (urban v. rural), age, hh income, gender, mode, purpose, and race. This sort of analysis would have been very slow and tedious either using Excel or ArcGIS Pro as pivot tables can only be so helpful.