PARKING TICKETS DW

Ryan Donovan

X16104269 Project B

A consulting business, Warehouse Consultants (WC), would like to show the Boston Transportation Department (BTD) the value of having a data warehouse to hold all of their information. It could be used to find insights to parking violations in order to generate more revenue. It could also sell this information to other Parking Associations as well to increase profit. WC will take the available data from BTD, along with other data sources to find some insights the combination of the data.

They interviewed people at BTD and also offered ideas to them to what things they should analyze. After this process, it was decided that they wanted integration of their parking tickets data with weather, gas prices, and restaurants in towns. WC focused on all the these though a true and more refined data warehouse could integrate vastly more data for insightful analysis. WC went out and gathered various types of data as will be discussed. They also did 4 case studies to show the power of a data warehouse used along with the Reporting software, Tableau.

- Data Sources

Seven different sources of data were either gathered, created/generated or captured through APIs or RStudio. First, the generated sources will be discussed. A temperature data file was created with integer temperatures from -50 to 150 degrees Fahrenheit in Excel. It was then exported as a CSV to be used later. A time data file was also generated in excel and exported as a CSV. A sample is shown below in Figure 1. The 4th column is a textual attribute that tells the time of day ((like "Early Morning", "Afternoon", etc.)

Time (Military)	Hour	Minute	Day Type
0:00	0	0	Early Morning
0:01	0	1	Early Morning
0:02	0	2	Early Morning
0:03	0	3	Early Morning
0:04	0	4	Early Morning
0:05	0	5	Early Morning
0:06	0	6	Early Morning
0:07	0	7	Early Morning
0:08	0	8	Early Morning
0:09	0	9	Early Morning
0:10	0	10	Early Morning

Figure 1

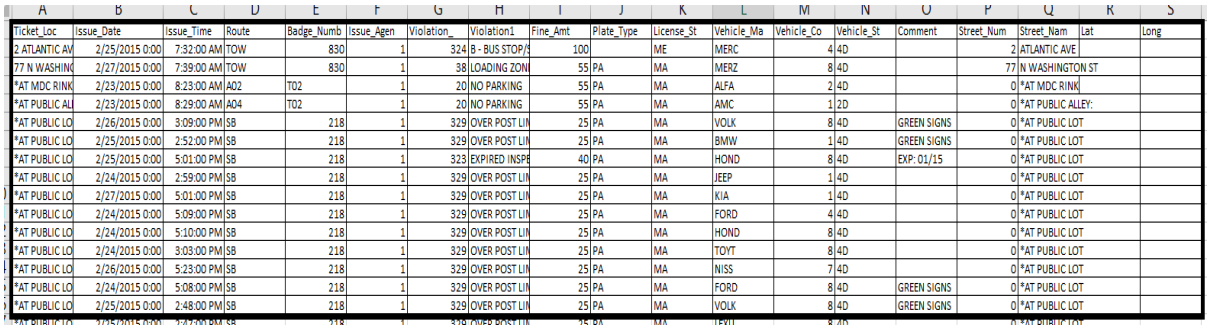
A Zip Code data set was the last of the generated data sets as shown in Figure 2. All the Zip Codes in Boston were found and matched to their town and city name. There was also a column of "Zones" that designated multiple Zip Codes (towns) to a Zone. This is used to show how flexible a data warehouse can be in making "designer" attributes. All zones were grouped in same. Notice that some Zip codes in the sample have 4 numbers when there should be 5. This is because Excel drops leading zeros but this will be fixed in the ETL process.

ZipCode	Town	Zone	City
2026	Brighton	1	Boston
2108	Boston	2	Boston
2109	Boston	2	Boston
2110	Boston	2	Boston
2111	Boston	2	Boston
2113	Boston	2	Boston
2114	Boston	2	Boston
2115	Boston	2	Boston
2116	Boston	2	Boston
2118	Roxbury	3	Boston
2119	Roxbury	3	Boston
2120	Mission Hill	3	Boston
2121	Dorchester	4	Boston
2122	Dorchester	4	Boston
2124	Dorchester	4	Boston

Figure 2

The BTD gave us a source for some of their data on the cityofboston.gov website (*Parking tickets issued 2/23/2015 - 3/1/2015*). A sample is shown below in Figure 3. You can see there are 19

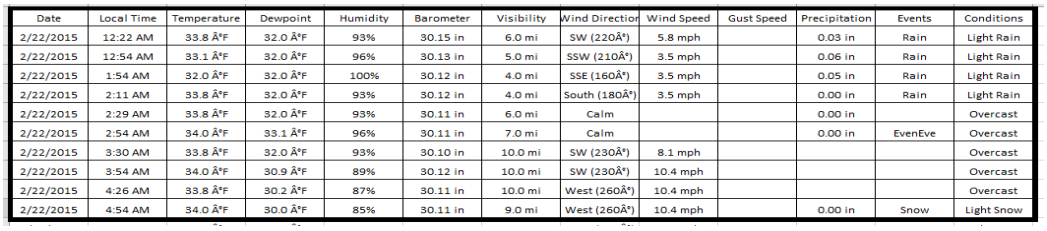
columns but we will be only be using some of them. Some of the columns had dirty data that was cleaned up in Google Refine as will be explained later.



Ticket_Loc	Issue_Date	Issue_Time	Route	Badge_Numb	Issue_Agen	Violation_	Violation1	Fine_Amt	Plate_Type	License_St	Vehicle_Ma	Vehicle_Co	Vehicle_St	Comment	Street_Num	Street_Nam	Lat	Long
2 ATLANTIC AV	2/25/2015 0:00	7:32:00 AM	TOW	830	1	324	B - BUS STOP/S	100		ME	MERC	4	4D		2	ATLANTIC AVE		
77 N WASHIN	2/27/2015 0:00	7:39:00 AM	TOW	830	1	38	LOADING ZON	55	PA	MA	MERZ	8	4D		77	N WASHINGTON ST		
*AT MDC RINK	2/23/2015 0:00	8:23:00 AM	A02	T02	1	20	NO PARKING	55	PA	MA	ALFA	2	4D		0	*AT MDC RINK		
*AT PUBLIC ALL	2/23/2015 0:00	8:29:00 AM	A04	T02	1	20	NO PARKING	55	PA	MA	AMC	1	2D		0	*AT PUBLIC ALLEY:		
*AT PUBLIC LO	2/26/2015 0:00	3:09:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	VOLK	8	4D	GREEN SIGNS	0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/25/2015 0:00	2:52:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	BMW	1	4D	GREEN SIGNS	0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/25/2015 0:00	5:01:00 PM	SB	218	1	323	EXPIRED INSPE	40	PA	MA	HOND	8	4D	EXP: 01/15	0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/24/2015 0:00	2:59:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	JEEP	1	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/27/2015 0:00	5:01:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	KIA	1	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/24/2015 0:00	5:09:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	FORD	4	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/24/2015 0:00	5:10:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	HOND	8	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/24/2015 0:00	3:03:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	TOYT	8	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/26/2015 0:00	5:23:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	NISS	7	4D		0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/24/2015 0:00	5:08:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	FORD	8	4D	GREEN SIGNS	0	*AT PUBLIC LOT		
*AT PUBLIC LO	2/25/2015 0:00	2:48:00 PM	SB	218	1	329	OVER POST LIN	25	PA	MA	VOLK	8	4D	GREEN SIGNS	0	*AT PUBLIC LOT		

Figure 3

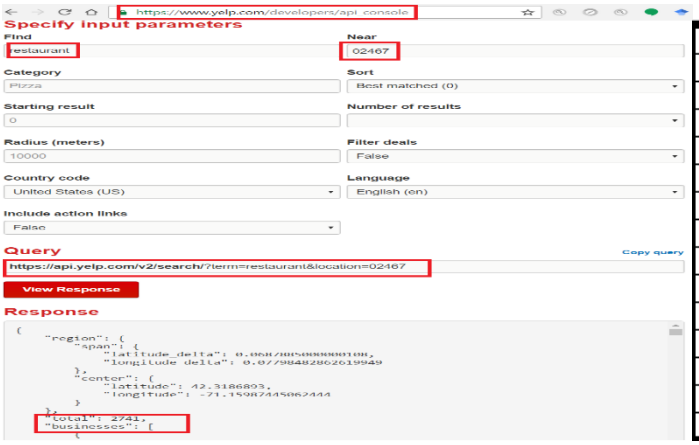
A semi structured weather data set was taken from weatherbase.com (Figure 4) for the same days that parking data covered. The grain of the parking ticket data was at the minute level and the weather data was roughly every ½ hour to hour. In a more robust data warehouse a company would pay for data at minute level but WC made some guesstimates and averaged out the time between measurements to fill in temperatures for the minute level. They assumed weather doesn't change a great deal in ½ to hour which is not great guesstimate but they were just trying to do a proof-of-concept.



Date	Local Time	Temperature	Dewpoint	Humidity	Barometer	Visibility	Wind Direction	Wind Speed	Gust Speed	Precipitation	Events	Conditions
2/22/2015	12:22 AM	33.8 Â°F	32.0 Â°F	93%	30.15 in	6.0 mi	SW (220Â°)	5.8 mph		0.03 in	Rain	Light Rain
2/22/2015	12:54 AM	33.1 Â°F	32.0 Â°F	96%	30.13 in	5.0 mi	SSW (210Â°)	3.5 mph		0.06 in	Rain	Light Rain
2/22/2015	1:54 AM	32.0 Â°F	32.0 Â°F	100%	30.12 in	4.0 mi	SSE (160Â°)	3.5 mph		0.05 in	Rain	Light Rain
2/22/2015	2:11 AM	33.8 Â°F	32.0 Â°F	93%	30.12 in	4.0 mi	South (180Â°)	3.5 mph		0.00 in	Rain	Light Rain
2/22/2015	2:29 AM	33.8 Â°F	32.0 Â°F	93%	30.11 in	6.0 mi	Calm			0.00 in		Overcast
2/22/2015	2:54 AM	34.0 Â°F	33.1 Â°F	96%	30.11 in	7.0 mi	Calm			0.00 in	EvenEve	Overcast
2/22/2015	3:30 AM	33.8 Â°F	32.0 Â°F	93%	30.10 in	10.0 mi	SW (230Â°)	8.1 mph				Overcast
2/22/2015	3:54 AM	34.0 Â°F	30.9 Â°F	89%	30.12 in	10.0 mi	SW (230Â°)	10.4 mph				Overcast
2/22/2015	4:26 AM	33.8 Â°F	30.2 Â°F	87%	30.11 in	10.0 mi	West (260Â°)	10.4 mph				Overcast
2/22/2015	4:54 AM	34.0 Â°F	30.0 Â°F	85%	30.11 in	9.0 mi	West (260Â°)	10.4 mph		0.00 in	Snow	Light Snow

Figure 4

The last 2 datasets discussed are unstructured data. The first got restaurant counts for all the zip codes for all the dates covered by using the Yelp API in Figure 5A. Each Zipcode was entered and then the restaurant count was recorded. Figure 5B shows the data file created. Though the grain is at day level it will be transformed to minute level because WC assumes the restaurant count of a neighborhood doesn't change too quickly. This data can be used to see if there is a connection to the amount of tickets given out to the number of restaurants in a zip code, zone, etc.

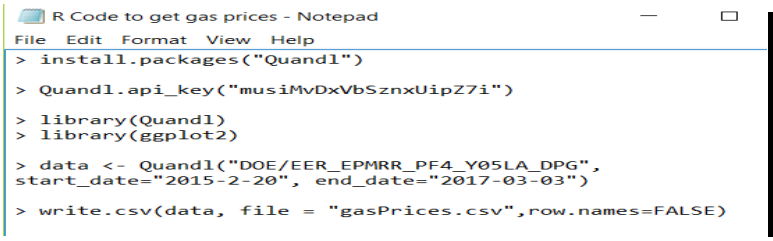


Date	ZipCode	rest count
2/22/2015	2026	615
2/22/2015	2108	1347
2/22/2015	2109	992
2/22/2015	2110	1056
2/22/2015	2111	2183
2/22/2015	2113	456
2/22/2015	2114	1194
2/22/2015	2115	2350
2/22/2015	2116	1404
2/22/2015	2118	915
2/22/2015	2119	1818
2/22/2015	2120	2149
2/22/2015	2121	802
2/22/2015	2122	420

Figure 5A

Figure 5B

Figure 6A shows code used in R gathered to gather average gas prices from Quandl. Figure 6B shows the CSV file generated. It is data for Los Angeles but we will be using it for Boston. In robust warehouse, you would have to pay money for that data and we will be using LA data for the proof-of-concept. The date format is different than other data sources so it will be transformed later in the ETL process.



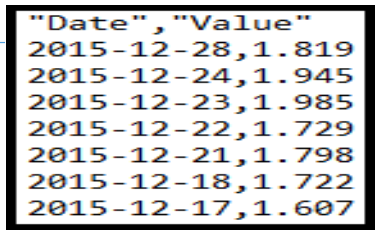


Figure 6A Figure 6B

- Schema and Architecture

A star schema dimensional model, paraded by Kimball, was chosen because of the ease of set up and for the ease of use by BTD and because it has been used by in many data warehouses successfully for years. A snowflake schema can offer some performance improvements but would take hits on the ease of use. The ultimate end goal is for the business users to easily use the deployed cube with Tableau or other Analysis Services. Figure 7 shows the schema for the database that will hold the data warehouse. All of the dimensions have primary keys that will connect to the same named foreign keys in fact table. The "fact_ticket_amount" will be an additive fact but the other 2 measures "Gas_Price" and "YelpRestaurantCount" will need to be averaged out for their record counts in queries. This will be shown in the reporting section later on. A staging area will be first used to clean data and then it will be transported to the data warehouse. Then it will be put into a cube that can be used for analysis and reporting.

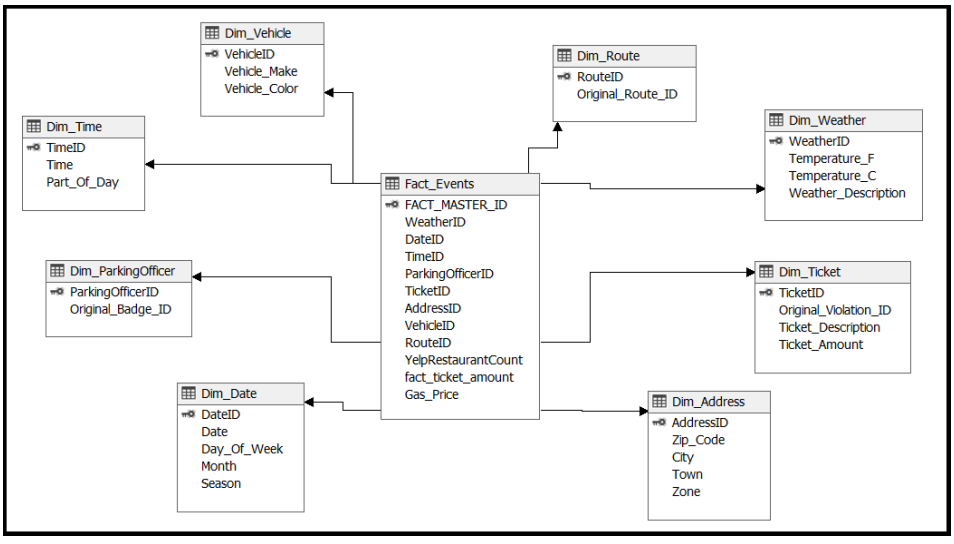
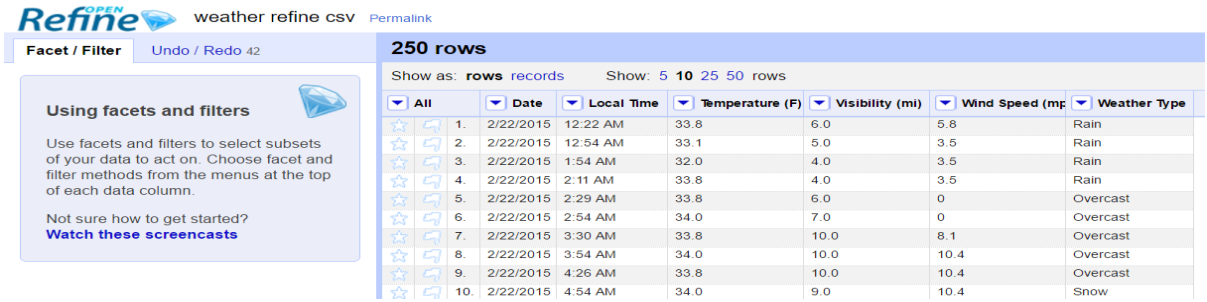


Figure 7

- ETL process

Most of the ETL process was done in SSIS but there was some pre-cleaning of data with Google Refine for ease of use and to show to BTD that it could be used as a tool if they decided to go

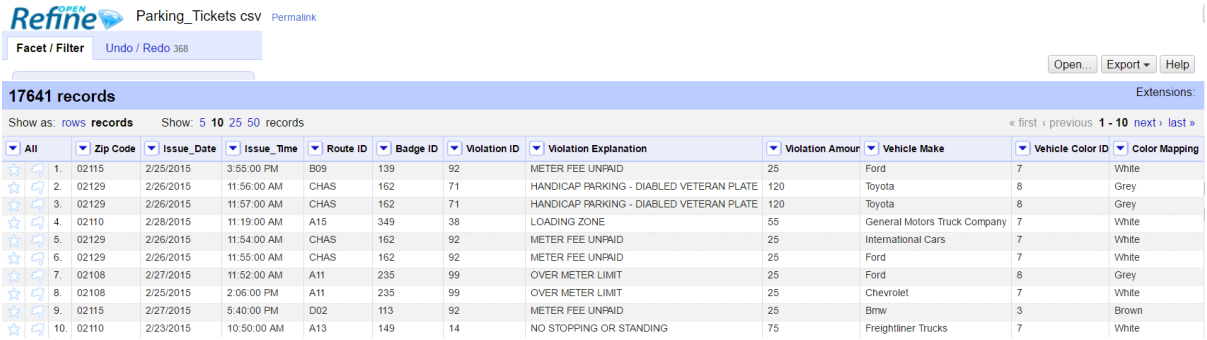
the data warehouse route. Refine was used to delete unwanted columns like Visibility and Wind Speed (Figure 8). Events and Conditions from Figure 4 were combined to one column called Weather Type as seen in Figure 8. The code looked like the following: if(cells.['Events'].value == "" , cells['Conditions'].value,cells['Events'].value). It was very easy to remove leading and trailing whitespace and fix or remove the units in cells (see Figure 4 for Temperature cells). After cleaning the data in Refine, it was put in an Excel format so that data could be added at minute granularity. The closer a time was to a full row of data, then that data was copied to that row. For instance, if the time was 2:55 AM on 2/22/2015, it would have the same row data as 2:54 AM in Figure 8, but 3:13 AM would have 3:30 AM row data.



All	Date	Local Time	Temperature (F)	Visibility (mi)	Wind Speed (mp	Weather Type
1.	2/22/2015	12:22 AM	33.8	6.0	5.8	Rain
2.	2/22/2015	12:54 AM	33.1	5.0	3.5	Rain
3.	2/22/2015	1:54 AM	32.0	4.0	3.5	Rain
4.	2/22/2015	2:11 AM	33.8	4.0	3.5	Rain
5.	2/22/2015	2:29 AM	33.8	6.0	0	Overcast
6.	2/22/2015	2:54 AM	34.0	7.0	0	Overcast
7.	2/22/2015	3:30 AM	33.8	10.0	8.1	Overcast
8.	2/22/2015	3:54 AM	34.0	10.0	10.4	Overcast
9.	2/22/2015	4:26 AM	33.8	10.0	10.4	Overcast
10.	2/22/2015	4:54 AM	34.0	9.0	10.4	Snow

Figure 8

Refine was also used to clean up parts of the parking ticket data file (Figure 9). Trailing space and unwanted columns were removed as before. The vehicle makes were made more descriptive rather than abbreviations as seen in Figure 9. Some violation explanations had very similar names but were spelled slightly different, had different capitalization and grammar. These were fixed so only a unique copy of violation/ticket type was present.



All	Zip Code	Issue_Date	Issue_Time	Route ID	Badge ID	Violation ID	Violation Explanation	Violation Amour	Vehicle Make	Vehicle Color ID	Color Mapping
1.	02115	2/25/2015	3:55:00 PM	B09	139	92	METER FEE UNPAID	25	Ford	7	White
2.	02129	2/26/2015	11:56:00 AM	CHAS	162	71	HANDICAP PARKING - DIABLED VETERAN PLATE	120	Toyota	8	Grey
3.	02129	2/26/2015	11:57:00 AM	CHAS	162	71	HANDICAP PARKING - DIABLED VETERAN PLATE	120	Toyota	8	Grey
4.	02110	2/28/2015	11:19:00 AM	A15	349	38	LOADING ZONE	55	General Motors Truck Company	7	White
5.	02129	2/26/2015	11:54:00 AM	CHAS	162	92	METER FEE UNPAID	25	International Cars	7	White
6.	02129	2/26/2015	11:55:00 AM	CHAS	162	92	METER FEE UNPAID	25	Ford	7	White
7.	02108	2/27/2015	11:52:00 AM	A11	235	99	OVER METER LIMIT	25	Ford	8	Grey
8.	02108	2/25/2015	2:06:00 PM	A11	235	99	OVER METER LIMIT	25	Chevrolet	7	White
9.	02115	2/27/2015	5:40:00 PM	D02	113	92	METER FEE UNPAID	25	Bmw	3	Brown
10.	02110	2/23/2015	10:50:00 AM	A13	149	14	NO STOPPING OR STANDING	75	Freightliner Trucks	7	White

Figure 9

In staging area database I basically cleaned the data through making sure of data type matches, sending nulls and errors to files to be fixed independently, did data conversion, derived columns, and more of which will be shown in detail throughout the following. After the staging is done, the "cleaned" data is mapped to dimensional tables, a temporary fact table, and then to the final fact table to be used in data warehouse and cubes for analysis. I needed a temporary fact table to map data to fact table because I used an IDENTITY to make FACT_MASTER_ID for the fact table. You can't perform lookups when table has an IDENTITY primary key so the process was separated. The same was done for dimension as Staging tables for each table were created (Figure 10A) and then this data was mapped to their dimensional table counterparts (Figure 10B). Figure 10A shows the SQL code to create Staging Area and Figure 10B shows SQL code to generate the actual Data Warehouse. It also clearly separates the staging from the actual insertion of data to the Data Warehouse.

```
USE The_Staging_Area;

CREATE TABLE Stage_Weather(
        Temperature_Zone varchar(11),
        Temperature_F int,
        Temperature_C int,
        Weather_Description varchar(50)
);

CREATE TABLE Stage_Date(
        [Date] date,
        Day_Of_Week varchar(9),
        Month varchar(9),
        Season varchar(10)
);

CREATE TABLE Stage_Time(
        [Time] time(0),
        Part_Of_Day varchar(20)
);

CREATE TABLE Stage_Parking_Officer(
        Original_Badge_ID varchar(20)
);

CREATE TABLE Stage_Ticket(
        Original_Violation_ID int,
        Ticket_Description varchar(50),
        Ticket_Amount money
);

CREATE TABLE Stage_Address(
        Zip_Code varchar(5),
        City varchar(50),
        Town varchar(50),
        Zone int
);

CREATE TABLE Stage_Vehicle(
        Vehicle_Make varchar(50),
        Vehicle_Color varchar(50)
);

CREATE TABLE Stage_Route(
        Original_Route_ID varchar(20)
);
```

Figure 10A

```
CREATE DATABASE Ticket_Weather_Yelp_DataWarehouse;
USE Ticket_Weather_Yelp_DataWarehouse;

CREATE TABLE Dim_Weather(
        WeatherID int IDENTITY (1,1) PRIMARY KEY,
        Temperature_F int,
        Temperature_C int,
        Weather_Description varchar(50)
);

CREATE TABLE Dim_Date(
        DateID int IDENTITY(1, 1) PRIMARY KEY,
        [Date] date,
        Day_Of_Week varchar(9),
        Month varchar(9),
        Season varchar(10)
);

CREATE TABLE Dim_Time(
        TimeID int IDENTITY(1, 1) PRIMARY KEY,
        [Time] time(0),
        Part_Of_Day varchar(20)
);

CREATE TABLE Dim_ParkingOfficer(
        ParkingOfficerID int IDENTITY(1, 1) PRIMARY KEY,
        Original_Badge_ID varchar(20)
);

CREATE TABLE Dim_Ticket(
        TicketID int IDENTITY(1, 1) PRIMARY KEY,
        Original_Violation_ID int,
        Ticket_Description varchar(50),
        Ticket_Amount money
);

CREATE TABLE Dim_Address(
        AddressID int IDENTITY(1, 1) PRIMARY KEY,
        Zip_Code varchar(5),
        City varchar(50),
        Town varchar(50),
        Zone int
);

CREATE TABLE Dim_Vehicle(
        VehicleID int IDENTITY(1, 1) PRIMARY KEY,
        Vehicle_Make varchar(50),
        Vehicle_Color varchar(50)
);

CREATE TABLE Dim_Route(
        RouteID int IDENTITY(1, 1) PRIMARY KEY,
        Original_Route_ID varchar(20)
);

CREATE TABLE fact_temp(
        WeatherID int,
        DateID int,
        TimeID int,
        ParkingOfficerID int,
        TicketID int,
        AddressID int,
        VehicleID int,
        RouteID int,
        YelpRestaurantCount int,
        fact_ticket_amount money,
        Gas_Price money
);

CREATE TABLE Fact_Events(
        FACT_MASTER_ID int IDENTITY(1, 1) PRIMARY KEY,
        WeatherID int,
        DateID int,
        TimeID int,
        ParkingOfficerID int,
        TicketID int,
        AddressID int,
        VehicleID int,
        RouteID int,
        YelpRestaurantCount int,
        fact_ticket_amount money,
        Gas_Price money
);
```

Figure 10B

I created 2 SSIS packages to perform the ETL process: (1) “Populate Stage Tables.dtsx” and (2) “Populate Dimensions.dtsx” of which are shown below in Figures 11A-B. “Populate Stage Tables” populate the temporary staging tables and performs some extraction and transformations. “Populate Dimensions” loads/maps all the data from staging tables and a few CSV files to the Dimension and Fact Table. Also after the dimensions and are populated, the staging tables are cleared in the last data flow step in order to save space and for new data to be entered at a later time into the folder that SSIS grabs the data files from. Both packages take one click each to be enacted so the ETL is almost fully automated once it has the data sources.

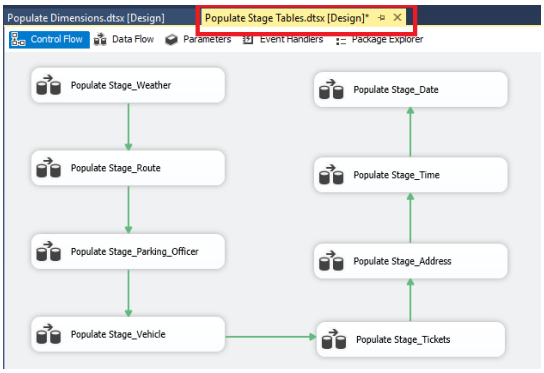


Figure 11A

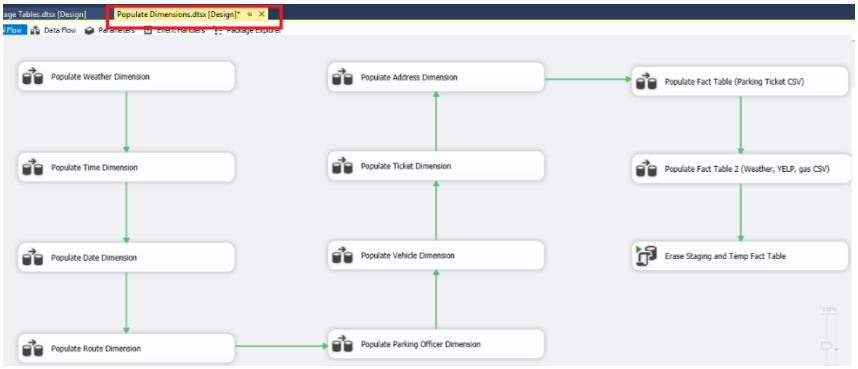


Figure 11B

The data flow for weather staging table is pictured below in Figure 12A. First, we only take one rows from the data source (CSV_Weather in Figure 12B) and then the ETL process checks for uniqueness. Uniqueness is done at beginning (if possible) to reduce the reprocessing of same data over and over. They then converted the data type from SSIS's automatic "string [DT_STR] (length 255)" to our desired type of DT_STR (50). If there are any errors or rows with nulls they are sent to separate files each to be dealt with later on. Figure 12C below shows a folder where all our error and rows with null files will be sent. An example of checking for nulls is also shown as well in Figure 12D. This error and null process will be repeated in most of our Data Flow tasks in the staging area so it won't be shown later on. Next, a join key (equal to 1) is created for Weather Type data flow and is sorted because we will be joining it with data from a Temperature File. Sorting happens because you can't do a join in SSIS otherwise. A similar process is done with "CSV_Temperature" (Figure 12B). Unique temperatures are taken and then temperature is converted to a 4-byte integer (DT_I4). Errors and nulls are taken care of as done with Weather_CSV file. Following, we derive Celsius equivalent and "Temperature Zones". The Celsius derivation is shown below in Figure 12E. "Temperature Zones" is a description of different ranges as in following: Freezing, Cold, Cool, Average, Warm, Hot, and Blazing Hot (also in Figure 12 E. Next, we convert temperature zone to our desired data type of DT_SR (11) because we won't have any textual attributes bigger than 11 since they derived those themselves. A join key is then created and sorted as done previously for the temperature data flow. A full outer join is done to get all the possible permutations from each row so all temperature can have all the possible "Weather Type" values. In this data warehouse, we had 7 unique Weather Types and 201 Temperatures which give us (7 * 201) 1407 possible combinations. Note that we will probably never get snow when its 90°F but for ease of creation we kept in the warehouse. It also takes up such small amount of disk space it will not affect performance. After the join, a lookup is performed to check if the rows are already in the staging table so there aren't repeated rows. Rows not in the staging are then inserted into the Weather Staging table to be processed later. A For Each Folder for multiple files could be implemented in this but was left out because this is a proof-of-concept and it could be easily implemented in a more robust warehouse.

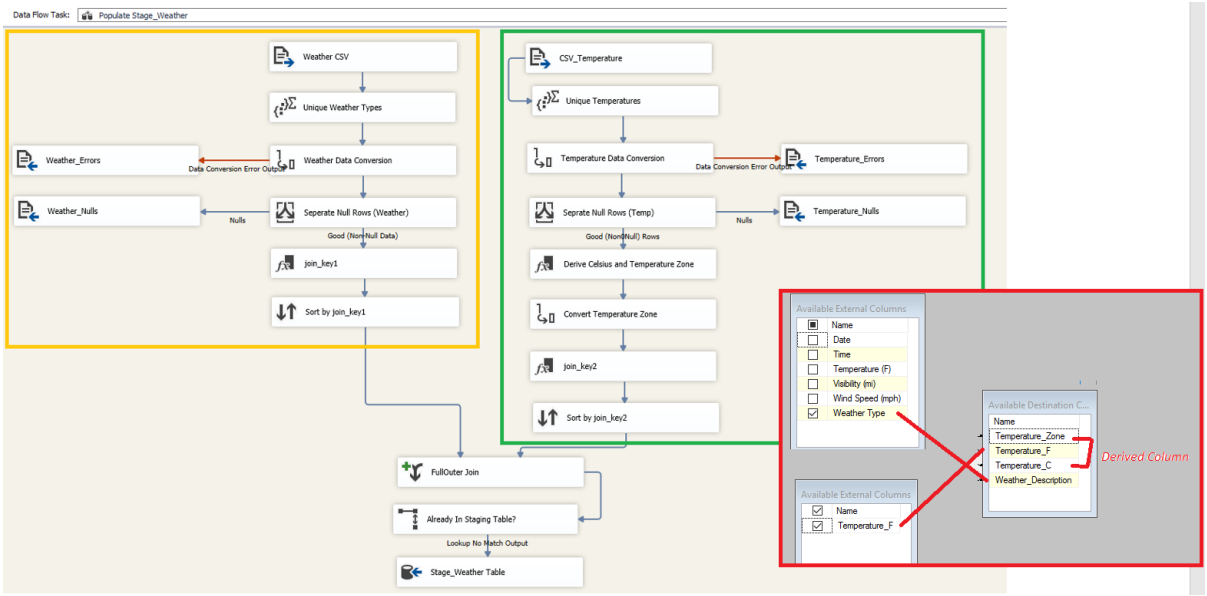


Figure 12A

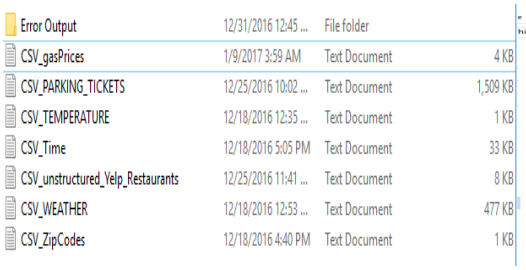


Figure 12 B

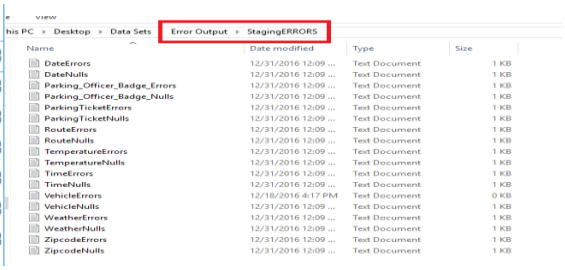


Figure 12C

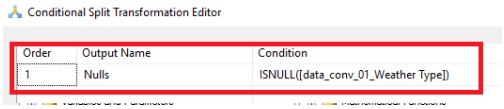


Figure 12D

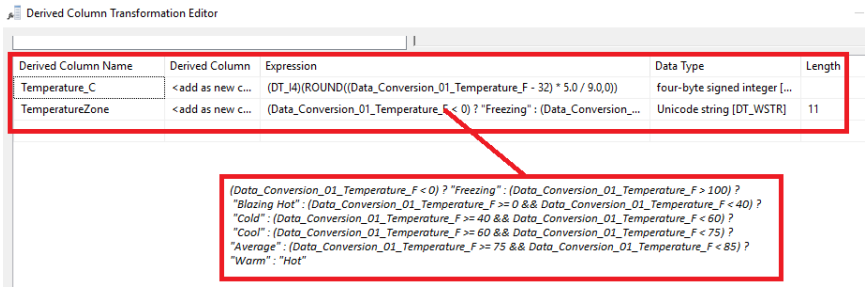


Figure 12E

The Route staging table data flow is a rather simple process (Figure 13). It first takes the Route ID from Parking Ticket CSV and then finds the unique values. It then converts it to a DT_SR(20) as it isn't expected for them to be over 20 characters. It then checks for nulls and errors as done previously. As done before, a lookup checks to see if rows are in staging table and inserts them if they are not in the table. The Route _IDs from source file are mapped to Original_Route_ID as shown in Figure 13 so original IDs are kept even though new unique ID will be created later. In a more robust data warehouse we would gather more info about a route, like it's location, time of

creation/ending, length, etc. We could use time of creation and ending attributes gathered in case we had 2 or more repeated Original_Route_IDs. That way when creating unique ID later it could tell which route it was referring to by time declarations and possible location info. The Original Route IDs were only able to be gathered and will be assumed to be unique because only 9 days of data was gathered and unlikely to change.

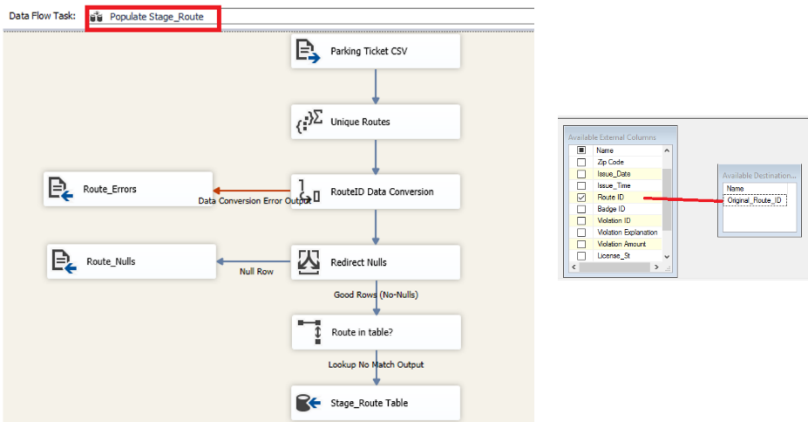


Figure 13

The parking officer staging table is very similar to the route table process (Figure 14). The Badge ID is taken from Parking Ticket CSV and then the unique values are taken. It is then converted to a DT_STR (20) as we again don't expect over 20 characters. We redirect rows with nulls or various rows to files. We check if these Badge IDs are in the Parking Officer Table and if not we insert them. Note these Badge IDs from CSV are mapped an Original_Badge_ID column to clarify that these are original IDs as we did with route. Also, we only had data for Badge ID but it would be useful in a real data warehouse to gather the Parking Officer's name, address, DOB, SSN in order that we have unique Parking Officer ID for each person. We are assuming that all Original Badge IDs from the data are unique because we have a small-time span.

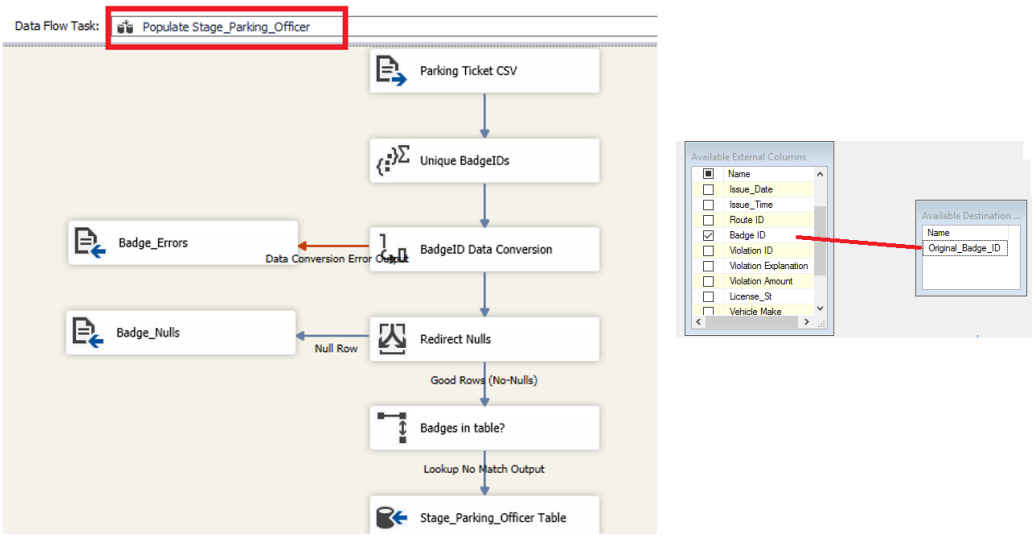


Figure 14

Populating the vehicle staging table was more complex than the previous 2 staging tables as shown in Figure 15. We sourced the vehicle make and vehicle colors columns from the Parking Ticket CSV. We then converted both to their correct datatypes (DT_STR (50) for both) and then redirected errors and null rows as done previously. We then did a conditional split of the 2 columns

because we wanted to do a full outer join as we did for weather type and temperature data earlier. We had 9 unique colors and 60 unique car makes in this file so 540 (9*60) rows were created. Note we had to make join keys and sort them as before so we could perform a full outer join. As before we check if these rows are in the staging table and then we insert them if they are not. An optimization of this process could be done by finding a data file with all the makes of vehicles worldwide and then join them with all the possible colors for cars to get a dimension for all possible vehicle combinations. Thusly, we could separate this into another SSIS package that we wouldn't have to process that often. More attributes could be added with more info like mpg, cylinder amount, electric/gas, and more to get more insight in our Analysis phase of the Data Warehouse.

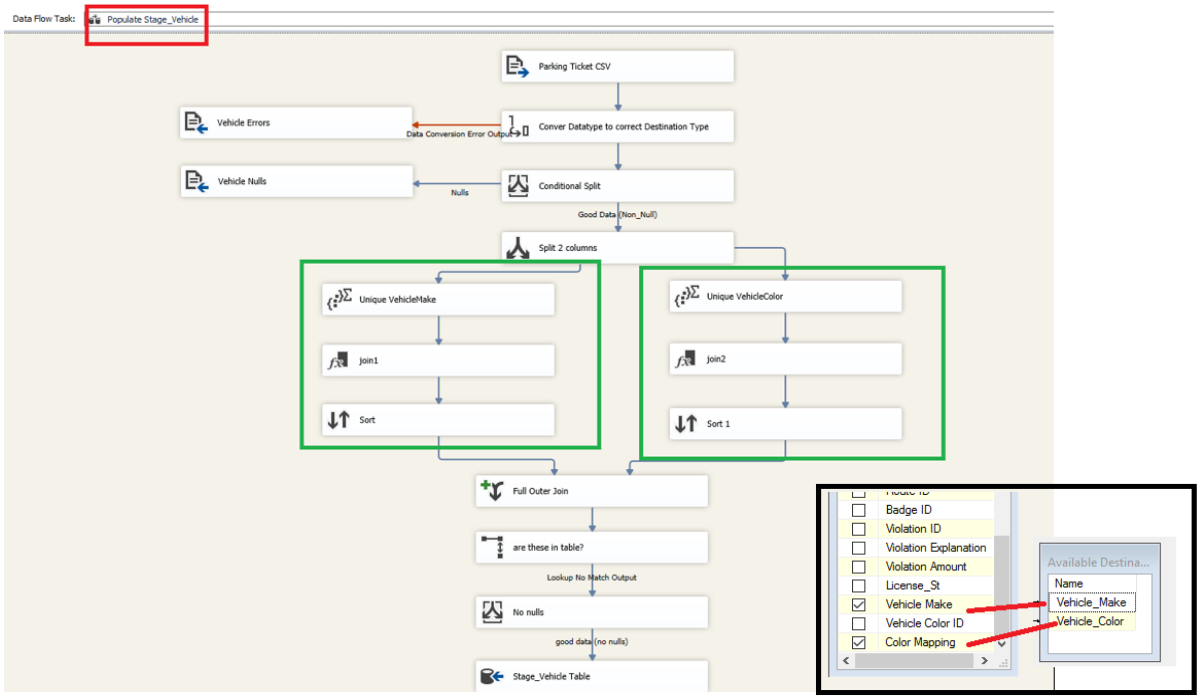


Figure 15

Three columns are taken form Parking Ticket CSV to populate the Ticket staging tables (Figure 16) (Violation ID, Violation Explanation, and Violation Amount) and unique rows are found. Data conversions are performed (ID to DT_I4, Explanation to DT_STR (50) and Amount to DT_CY) and the null and errors are redirected as before. A lookup is then performed to see if these rows are in the staging table and are inserted if not.

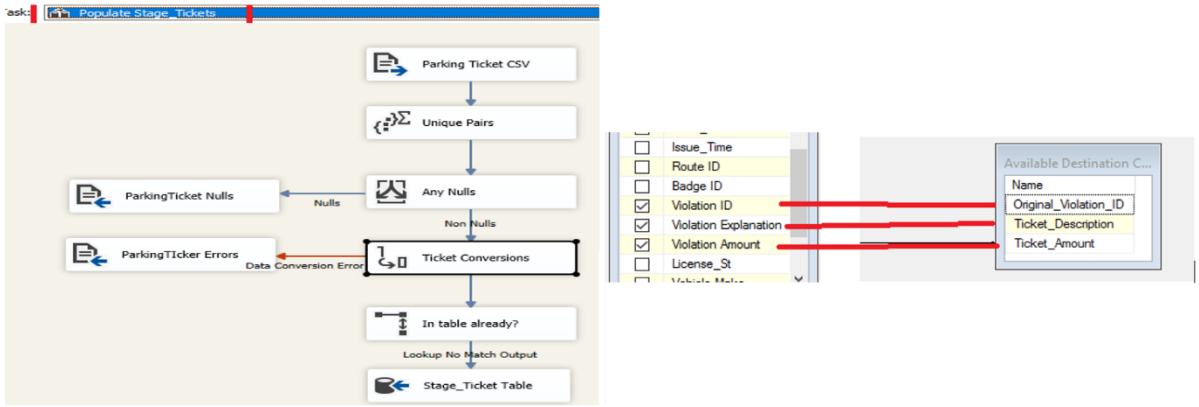


Figure 16

In the population of address staging table in Figure 17A, 4 columns were taken from the CSV_Zipcodes file and they are converted their correct datatypes (ZipCode int, Town DTSTR(50), ZoneDT_I4, and City DT_STR (50)). ZipCode is an 'int' because we want to direct ones that are not numbers to error file. Other errrors from other columns are also redirected. ZipCode is then converted to its destination data type of DT_STR (5) because all zip codes must be of length 5. A new column is then derived from zipcodes as shown in Figure 17B to convert the original zip codes in data file that have only 4 numbers to 5. This is needed when zipcode begins with 0 ("02132" might be represented as "2132" so it needs to have a "0" added to beginning). Unique rows are now found once again to reduce superfluous processing. The rows with nulls are now redirected to correct file. A lookup is performed again to see if the rows are not in staging table and inserts data accordingly. We had data for street addresses but the grain of our unstructured data for restauarant counts a was at ZipCode/Town (zipcode represents town) granularity so for ease of use in warehouse, we only made address's lowest granularty at ZipCode level. A more robust data warehouse could use street address if were able to get data at address level for restaurants.

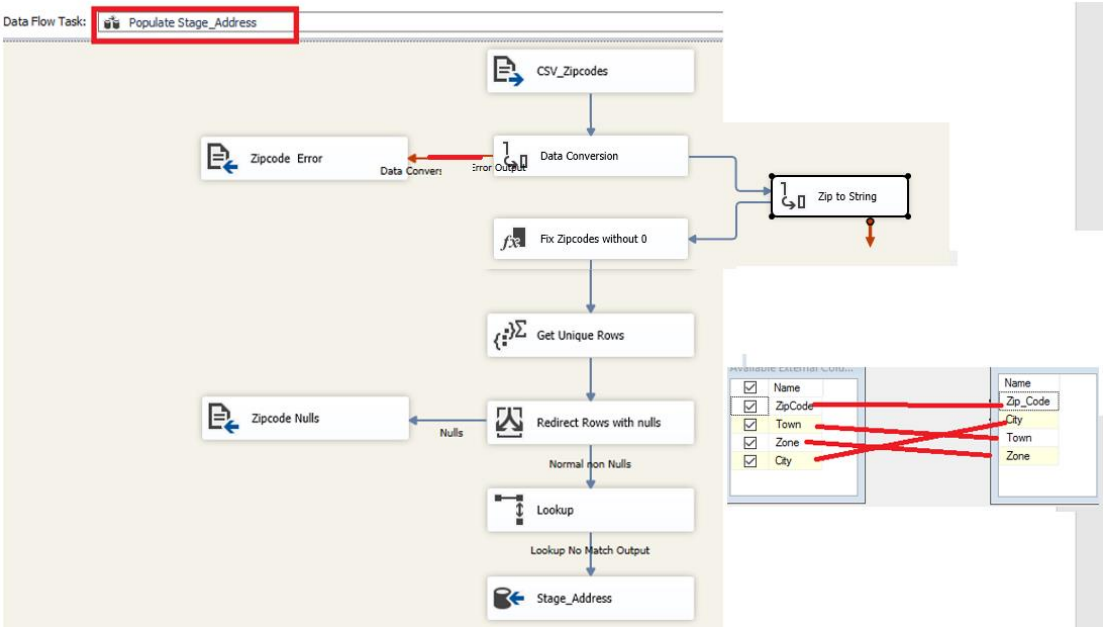


Figure 17A

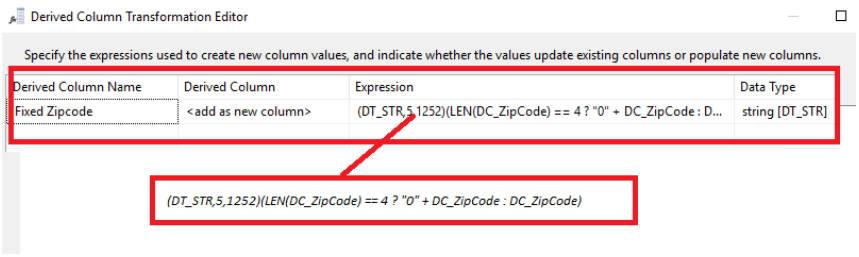


Figure 17B

The 2 columns, "Time (Military)" and "Day Type" are taken from CSV_Time file in order to populate the time staging table (Figure 18). They separated time from dimension table in order to save space because there would be a lot of unneeded space used for the date dimension if time was included in it. If the dimensions were combined there would be 1440 (60min/hr*24hr) as much space used up for that table. Firstly, we converted data types to DT_DBTIME2 for Time (Military) and to DT_STR (20) for Day Type. Errors are then directed to error files. Unique rows are then taken

and nulls are redirected to null file. Once again, a lookup is performed to see if data already in staging table.

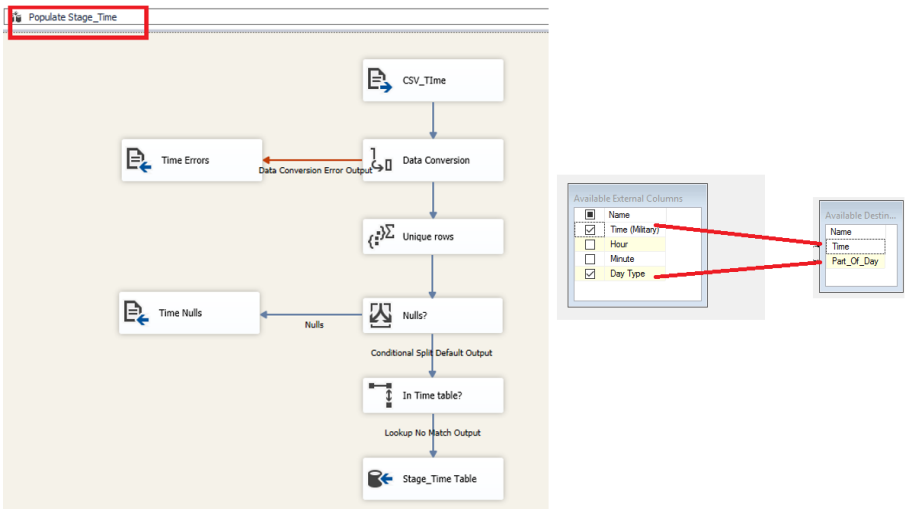


Figure 18

The Date staging table has 4 columns, of which 3 are derived from Date column from the CSV_Weather file. Figure 19A shows the overall process. The Date column is taken from weather and unique dates are taken. The Date is then converted to a DT_DBDATE data type and errors are directed to an error file. Nulls are then redirected as well to null file. “Day_Of_Week”, “Month” and “Season” are derived as shown in Figure 19B. Once again, the lookup is performed to see if rows are in staging table and then are inserted into table if not.

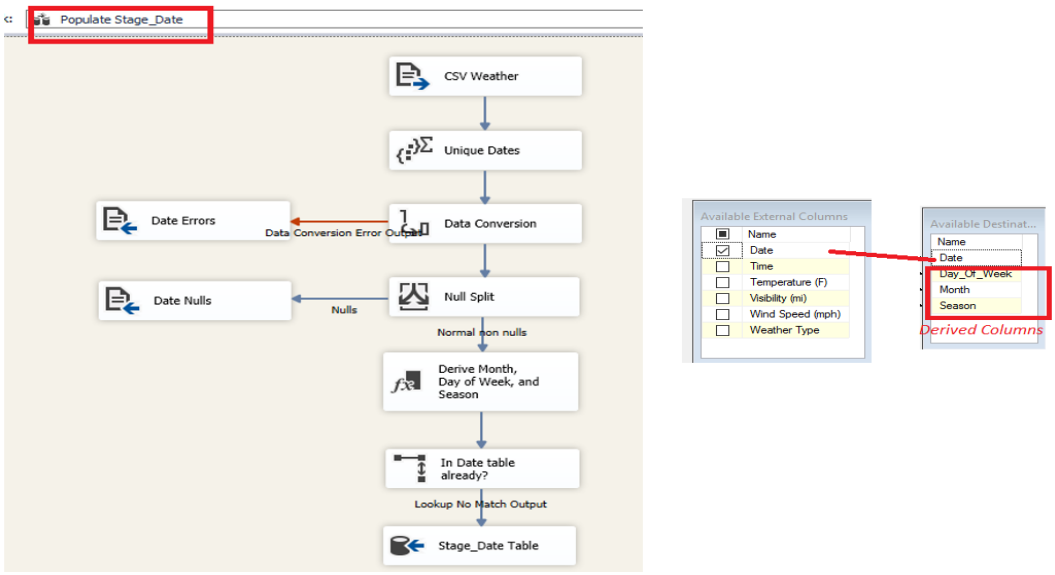


Figure 19A

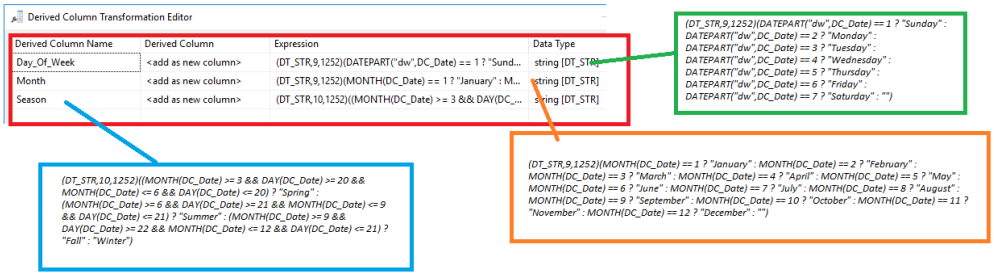


Figure 19B

The process for Package 1 in populating the Staging Area is now complete. The following will outline the process in 2nd Package that populates the dimensional and fact tables. The population of the dimensions is trivial as the exact same process is used for all. Figure 20 shows the process for populating the Weather Dimension as an example. All the columns from the Weather staging table are used to be sent to weather dimension table. A lookup is performed to see if the unique rows in staging are in Dimension. If not in dimension, they are inserted, else they are ignored. When inserted an IDENTITY column is created in Dimension table to create a unique integer ID (WeatherID for this one). There is no error or null checking needed as this was already taking care of in the staging process of first Package.

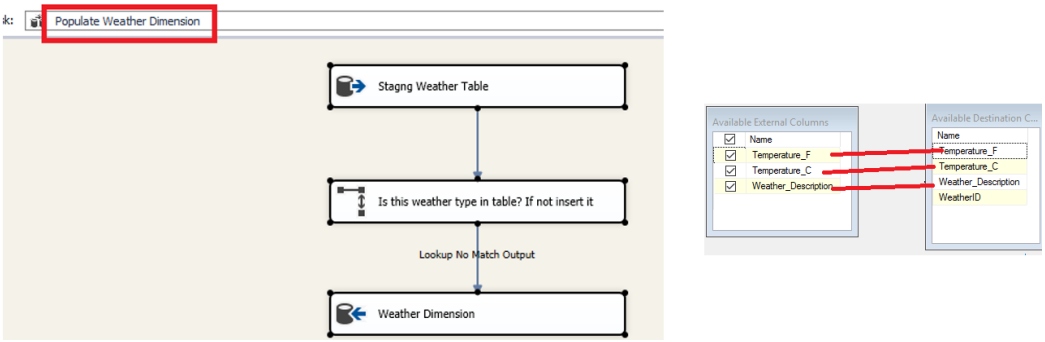


Figure 20

There were two major steps to populating the fact table. First step was mapping all the information form the Parking Ticket CSV to a Temporary Fact table as shown in Figure 21. We had a temporary one because the data flow task would be less convoluted and IDENTITY column problem stated earlier in the report. All the columns were selected from Parking Ticket source file and they were given all the right data types in order to perform lookups. Rows with nulls or errors from file are simply skipped because they would have been caught in first staging package. Zip codes that were 4 digits were fixed to make them 5 and then it was converted to a DT_STR (5). The next 3 steps performed lookups to get the corresponding Address ID, Date ID, Parking Officer ID, and Route ID. Next the Ticket ID was captured and its corresponding ticket amount was as well. The Time ID and Vehicle ID were then looked up as well. A lookup was then performed to see if these rows were in the temporary fact table and then they were inserted into it if they weren't already in it.

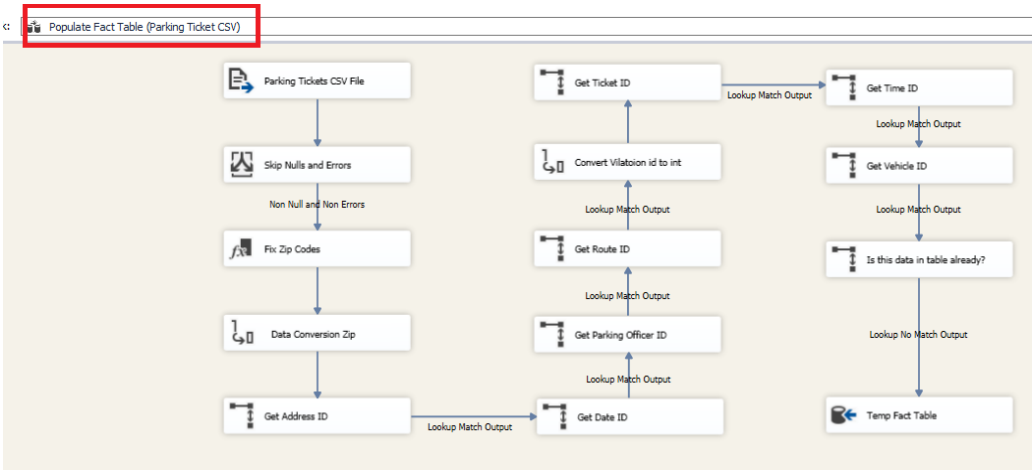


Figure 21

All the data from temporary fact table was merged with weather, yelp and gas prices data before finally being inserted into fact table if it was unique data as in Figure 22. In the pink box, data from Weather CSV is used to find the Date ID and the Time ID of a weather event. Then temperatures are rounded to nearest integer so a lookup can be used to look up Weather ID by temperature and Weather Type. The Date ID and Time IDs gathered are ascendingly, as well as temporary fact equivalents. A left join is then performed to that gives each fact row its corresponding Weather ID by using Date ID and Time ID as a join key. Data from Yelp restaurant file is then used to lookup Address ID and Date ID corresponding to each of its rows. The Yelp data is sorted by Date ID and then left joined to the Fact_temp data flow. Now each fact row gets its corresponding yelp Restaurant count corresponding to the correct Zip Code and Date ID. The Gas Prices for each day are then mapped to their corresponding date. The date had to be first formatted by deriving a new column as shown below. The date from gas file was in form like following "YYYY-MM-DD" and we needed it in "MM/DD/YYYY". After it was transformed it was converted to DT_DBDATE format to match Date dimensions' format as in Figure 23. The corresponding Date ID was looked up and sorted. Another left join was performed to match the Gas Price of each day to its corresponding Date ID. Finally, a lookup was performed to see if each row is not in Fact Table. If wasn't already in the table, it was then inserted into table.

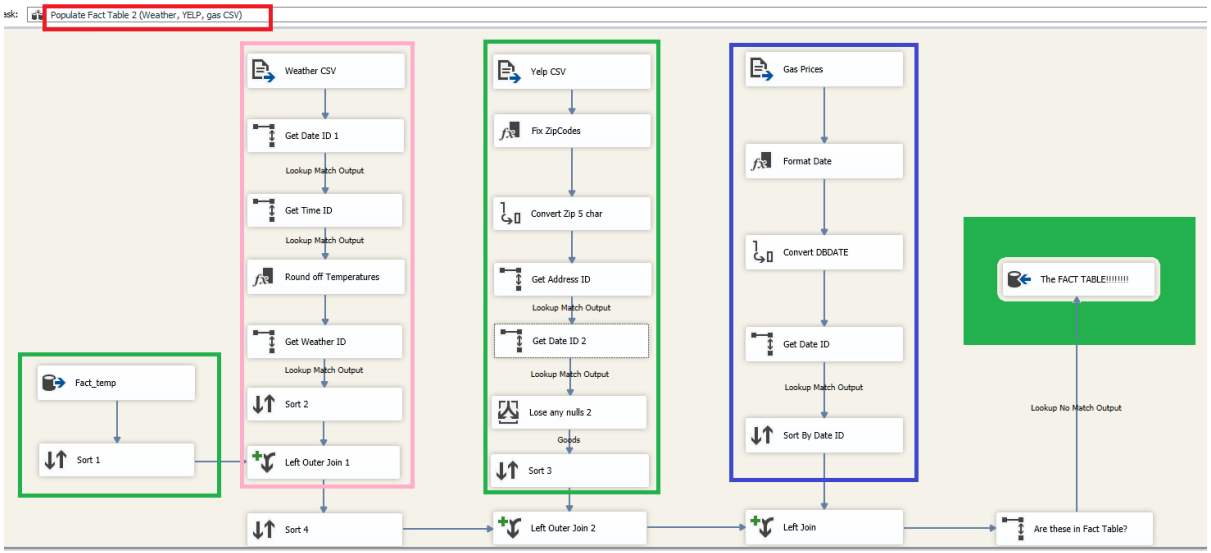


Figure 22

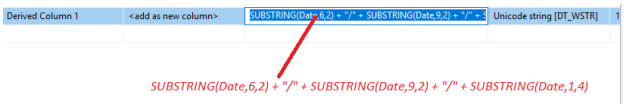


Figure 23

The final step in the control flow for the second package is to erase all the data from the staging tables and from the temporary fact table (Figure 24). So, if someone wanted to add new data to the warehouse you would simply add files with same names and formats to correct folder and just repeat the process. The overall process just needs a debugging of Package 1 and then Package 2 to populate all the dimensional and fact tables and to erase staging area afterwards.

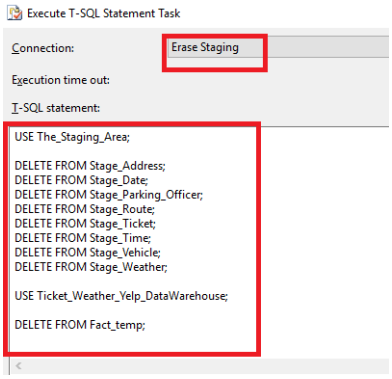


Figure 24

- Cube Deployment

The next step in the process was to deploy a cube in order to be used by Tableau, Excel pivot table or any other reporting service that can connect to SQL Server. I connected the cube to the database holding all the dimensions and fact table then deployed the cube. Figure 25 shows how the cube looks and shows the dimensions and fact table in the center.

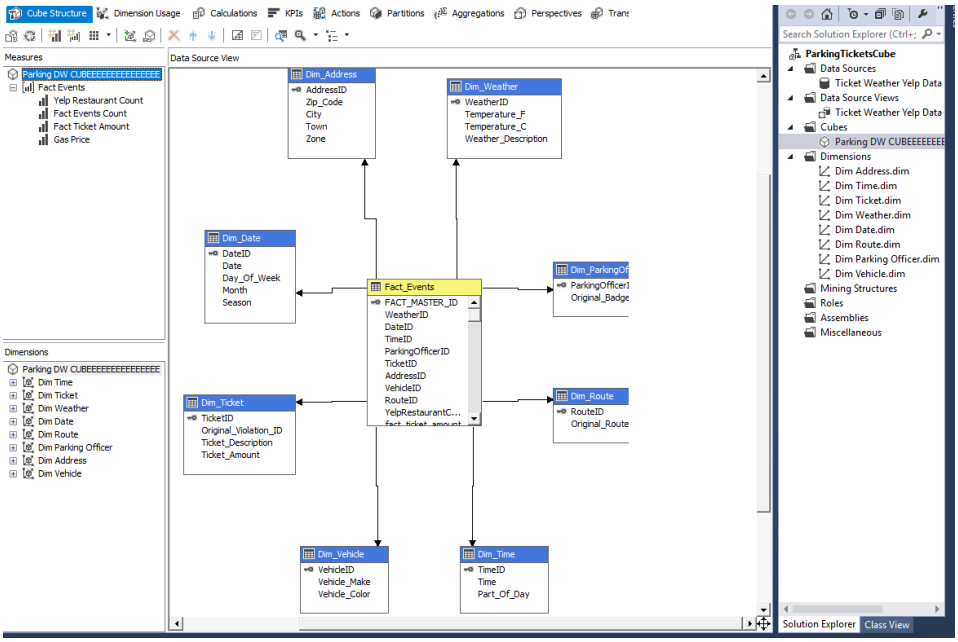


Figure 25

There were a few hierarchies that I defined in the process as well (Figures 25-26). The Zone hierarchy shows all the zip codes inside each designated zone. The ZipTown hierarchy is similar showing the zip codes and towns inside the city Boston.

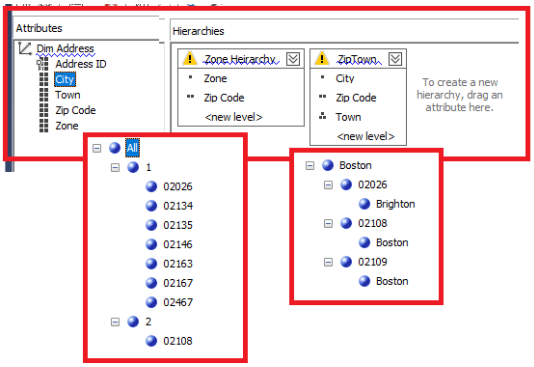


Figure 26

Below shows a Time Delineation hierarchy that shows the exact times inside each "Part of the Day". More hierarchies could be extended for added attributes at a later time if business required them.

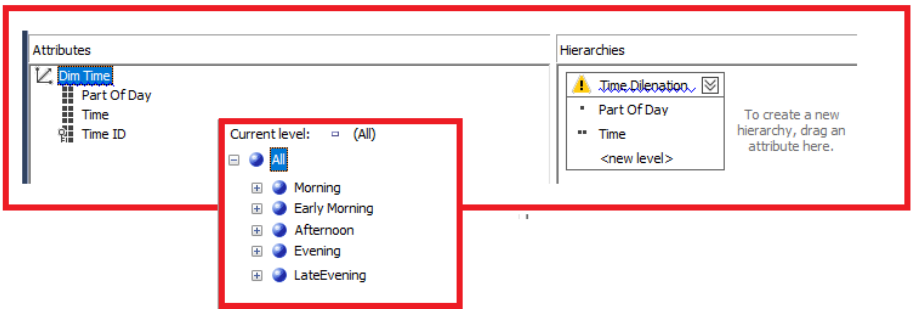


Figure 27

Analysis & Reporting (4 Case Studies)

Finally, Tableau was used to report some visualization for our case studies that the business might want to see. I connected to Tableau through my deployed cube as shown below in Figure 28.

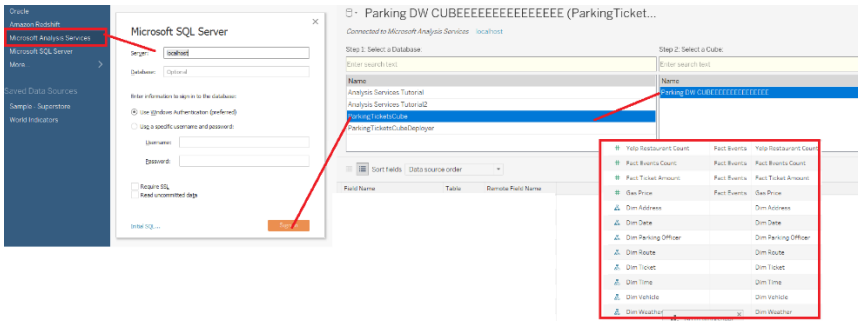


Figure 28

Case Study #1

Do different zones in the city of Boston have a higher ticket revenue over the day and at certain parts of the day? If so, which Zone is the best and which type of parking ticket is the best (brings in most revenue)? Figure 29A shows a query to help answer this question. Afternoon has the highest amount of revenue followed by Evening for all Zones. It suggests to put parking officers

on at those hours to generate more money. Also, zone 2 clearly has the most revenue generated by looking at the graph

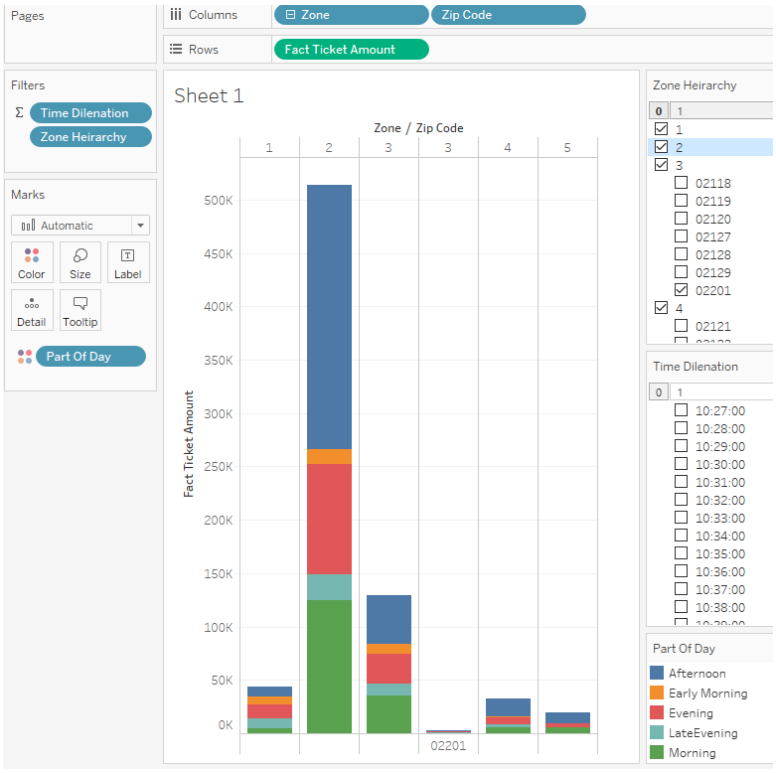


Figure 29A

The left graph in Figure 29B shows that the zip code 02116 generates most ticket revenue. The right graph shows that the “meter fee unpaid” ticket is the highest money earner for all Zones, even though it is not the highest priced ticket. Maybe it is because people don’t think it’s a major hit to the wallet to lose $25 so they are willing to risk getting a ticket. They might be more adverse for a $100 ticket.

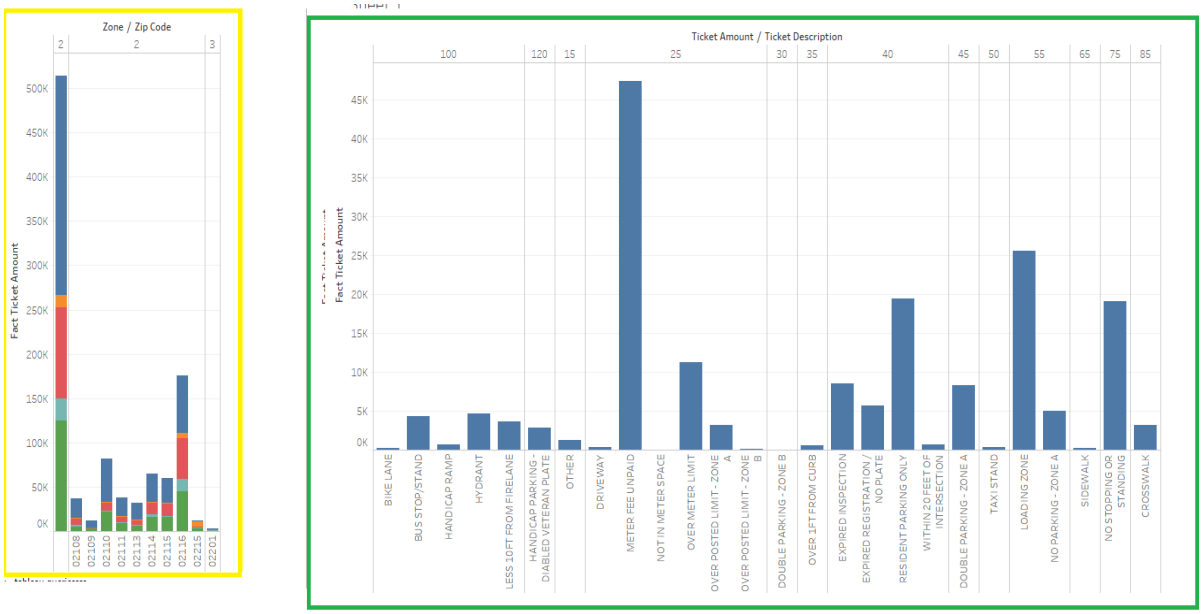


Figure 29B

Case Study #2

Is the amount of tickets per restaurant for a zip code a constant ratio? That is, are the tickets per restaurant different in different zip codes? I used Zip codes and divided the amount of tickets per average restaurant count to graph the query in Figure 30. There is no discernible pattern shown there but this could because different zip codes have different square areas. If the square area of each zip code was added as an attribute to the Address dimension we could get a better look at the results. If we got a better answer it might help BTD decide to put more meters or Parking Officers near restaurants. This could also be extended to drill down to breakfast, lunch and dinner places to be analyzed with different times of the day.

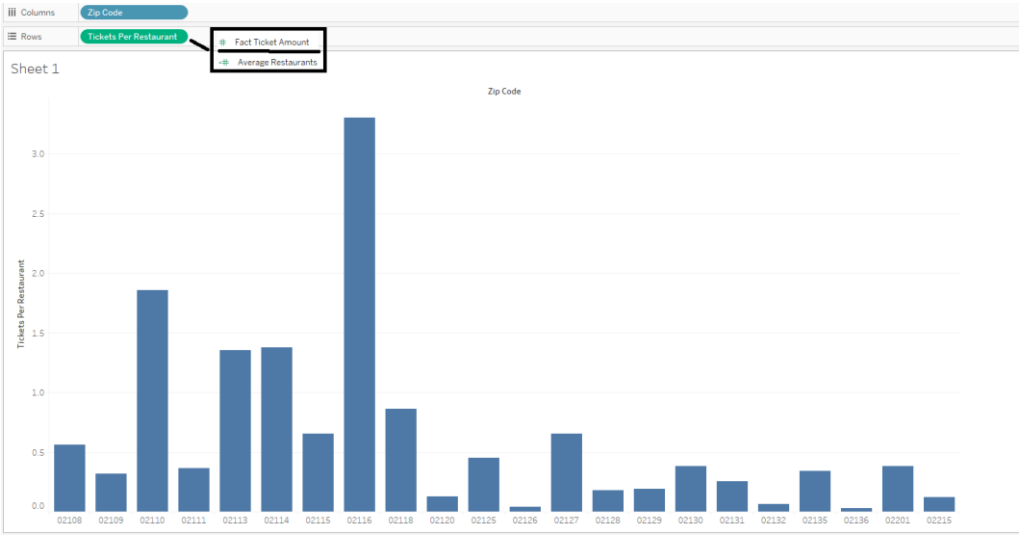


Figure 30

Case Study #3

Do different parking officers give out more tickets and is there a certain temperature they are given out more? As shown in Figure 31A, there seems to be a cluster of larger ticket amounts around 19-21 degrees Fahrenheit as seen by the big boxes in the graph below (bigger box means more tickets). 30 and 34 F both have semi clusters as well. We can conclude that we should have more officers giving out tickets on 18-22 F days. This is somewhat misleading as we only have 9 days of data. More data might help with better analysis and this could be enacted with a more robust warehouse if BTD chooses to make one. Figure 31A allows for a quick look at graph to see which Parking Officer has high ticket writing rates at different temperatures.

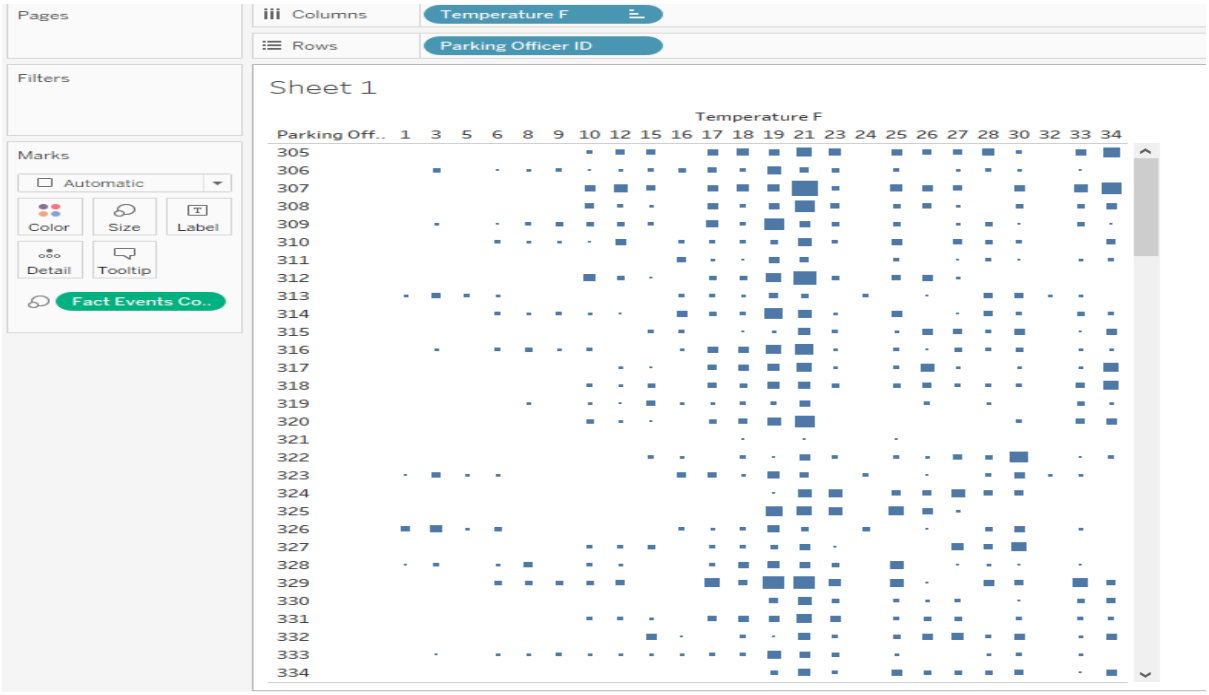


Figure 31A

Figure 31B shows a bubble chart showing the relative amounts of tickets each Parking Officer gave in all the data. The biggest bubbles show the larger ticket writers. Maybe they should be given overtime?

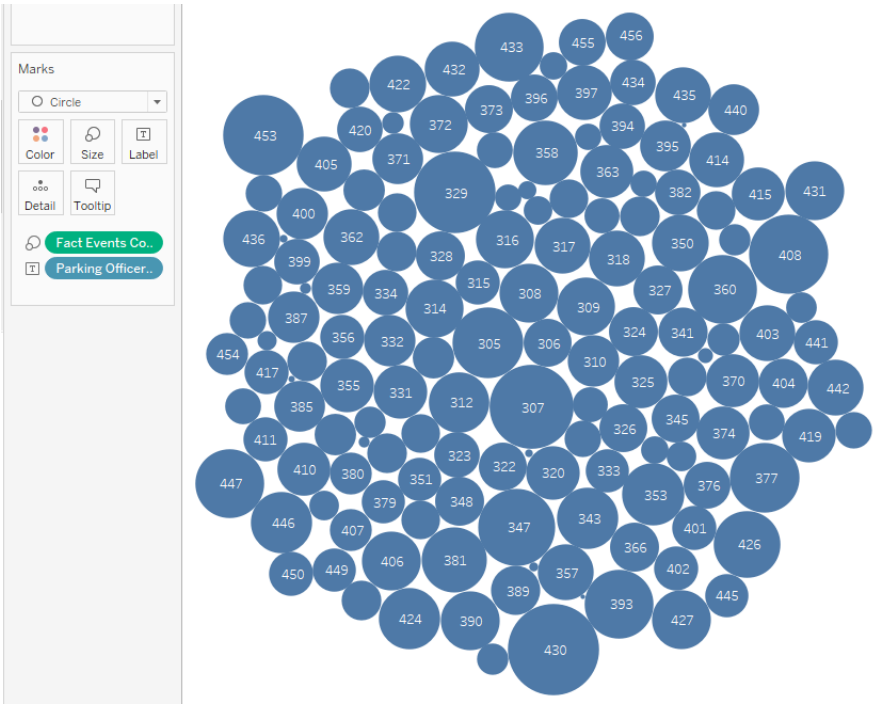


Figure 31B

Case Study #4

Do different color cars get more tickets? It seems that grey, black, and white have higher revenues and ticket amounts as compared to the other 5 colors in the study. This either suggests that certain

cars get ticketed more or that certain car color are bought at a higher rate than others. More attributes could be added to the Vehicle dimension to describe their buying rate to do a better analysis

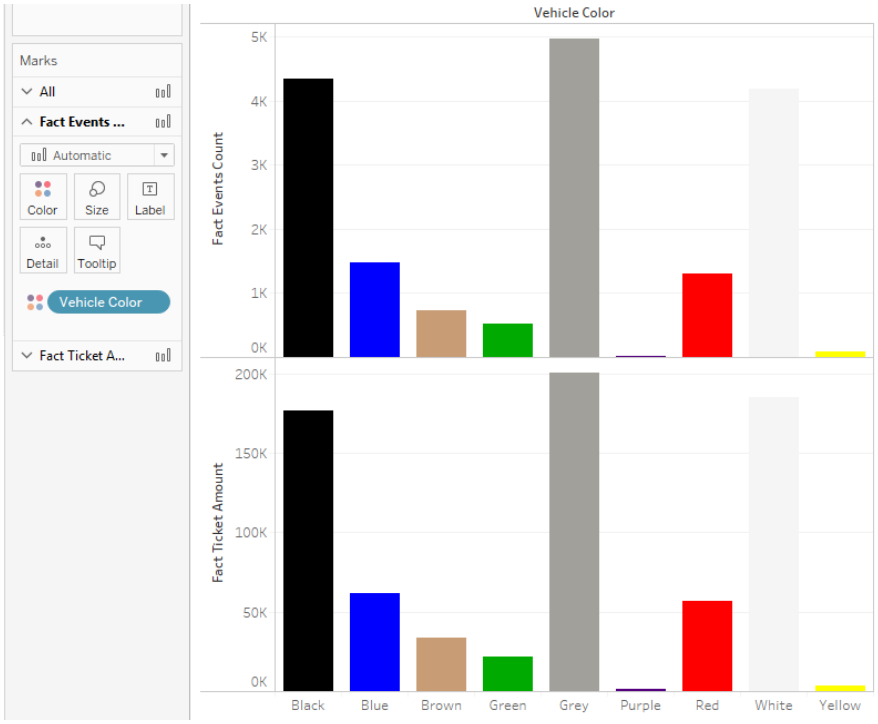


Figure 32

Bibliography

City of Boston.gov. (2015. *Parking Tickets* [CSV]. Retrieved from https://data.cityofboston.gov/dataset/Parking-Tickets/qbxx-ev3s

Inmon, W (2005) *Building the Data Warehouse* 4th ed. USA: Wiley Publishing, Inc.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J. and Becker B. (2008) *The Data Warehouse Lifecycle Toolkitit* 2nd ed. Indianapolis, Wiley Publishing, Inc.

Quandl (2017). Quandl.com. *Introduction: Overview to R.* Retrieved January 9, 2017, from https://www.quandl.com/docs/api?r#get-data

Thomas, A. (2013) 'API for Commodity Data', *Quandl BLOG,* 21 October, Available at: https://blog.quandl.com/api-for-commodity-data [Accessed 9 January 2017].

weatherbase. (2015). *Weather Data* [Tabular]. Retrieved from http://www.weatherbase.com/weather/weatherhourly.php3?s=90527&date=2015-02-