Penn State University

Great Valley Campus

Engineering Division

Data Specification for

NY Annualized Property Sales

Version 1.0

**1/20/2022**

**Table of Contents**

[Introduction 3](#_Toc457330103)

[Purpose 3](#_Toc457330104)

[Project Summary 3](#_Toc457330105)

[Requirements Definition 4](#_Toc457330106)

[Considerations 4](#_Toc457330107)

[Document Change Log 4](#_Toc457330108)

[Architecture Design 5](#_Toc457330109)

[Entity Relationship Diagram 5](#_Toc457330110)

[Data Dictionary 5](#_Toc457330111)

[Tables schemas 5](#_Toc457330112)

[Examples of values 5](#_Toc457330113)

[ETL Process 14](#_Toc457330114)

[ETL considerations 14](#_Toc457330115)

[ETL Process Flow with description 14](#_Toc457330116)

[Reporting System 27](#_Toc457330117)

[Conclusions 33](#_Toc457330118)

NY Annualized Property Sales  
Raditya Fahritama

# Purpose

The purpose of this project is to compare a data warehouse implementation with a Hadoop implementation.

# Project Summary

This paragraph is used to introduce the following subsections, which can be used for an executive level overview.

1. **Objectives**

The objective for this project is to implement automated ETL process that can run regularly and then make report of it. The project will try to find which sales is the most profitable among all of the provided areas. Which sales is the highest among all of the areas. And to find which neighborhood is the most profitable in sales. And to find which category is the most common of a building when they’re purchased. And then see if that category is profitable in sales. The project also will try to find the trends that occurred in the sales and how the prices evolved in different time for each of the locations.

1. **Scope**

This project is intended to analyze the performance of two different systems, which is data warehouse and Hadoop, with different architectures and same data and hardware, to see which is better to extract the knowledge of the given data.

# Requirements Definition

1. **Goals**

Goals of this project are:

- Run the given data in two different systems, which is data warehouse and Hadoop.

- Creating reports after running the process

- Analyze the systems. Specify the good and the bad of each system.

- Determine which system is more suitable for the given data.

- Get the answers for each business questions.

1. **Business Questions**

The business questions that the project will try to answer are:

- Which sales is the most profitable among all of the borough in 2011.

- What are the top 3 neighborhood of each borough to be invested in 2010.

- Which building category scores the highest in sales in 2011.

- Is there a trend in selling based on different time.

1. **Data Requirements**

Data that will be used for this project is provided. The data consists of some excel files that each of the file contains data of sales of the property based on annual reporting of each borough. The data provided has already pre-processed and reduced in size. The data contains 21 columns.

# 2. Architecture Design

## 2.1 Relational Data Warehouse

### Data Dictionary

A Data Dictionary is a document that describes the basic organization of a database. Typically, a data dictionary will contain a list of variables in the database as well as the assigned variable names and a description of each type of variable. The data dictionary should also include the values accepted for each variable and any helpful comment such as important exclusions and skip patterns. The data dictionary is used primarily for data analysis.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee Table** | | | | |
| Variable | Variable name | Variable type | Values | notes |
| Employee ID number | ID | Numeric | 001-900 |  |
| Employee name | ename | String | John Doe |  |
| Date of birth | DOB | dd/mm/yyyy | 1-31/1-12/1900-2011 |  |
| Sex | SEX | Numeric | 1 = male 2 = female |  |

### Tables schemas

The datasets given contain 21 columns. The columns are:

|  |
| --- |
| -BOROUGH |
| -NEIGHBORHOOD |
| -BUILDING CLASS CATEGORY |
| -TAX CLASS AT PRESENT |
| -BLOCK |
| -LOT |
| -EASE-MENT |
| -BUILDING CLASS AT PRESENT |
| -ADDRESS |
| -APARTMENT NUMBER |
| -ZIP CODE |
| -RESIDENTIAL UNITS |
| -COMMERCIAL UNITS |
| -TOTAL UNITS |
| -LAND SQUARE FEET |
| -GROSS SQUARE FEET |
| -YEAR BUILT |
| -TAX CLASS AT TIME OF SALE |
| -BUILDING CLASS AT TIME OF SALE |
| -SALE PRICE |
| -SALE DATE |

After analyzing the business questions and the columns given. It was decided that some columns will not be used in the architecture. The columns are:

|  |
| --- |
| -TAX CLASS AT PRESENT |
| -BLOCK |
| -LOT |
| -EASE-MENT |
| -BUILDING CLASS AT PRESENT |
| -APARTMENT NUMBER |
| -RESIDENTIAL UNITS |
| -COMMERCIAL UNITS |
| -TAX CLASS AT TIME OF SALE |
| -BUILDING CLASS AT TIME OF SALE |

And for the remaining columns. We tried to decide which columns are facts, and which columns are dimensions. We combine zipcode, borough, and neighborhood columns into one column. That is adressextension. This is the result:

|  |  |
| --- | --- |
| DIMENSIONS | FACTS |
| ADDRESSEXTENSION | TOTAL UNITS |
| BUILDING CLASS CATEGORY | LAND SQUARE FEET |
| ADDRESS | GROSS SQUARE FEET |
| YEAR BUILT | SALE PRICE |
| SALE DATE | FACTCOUNT |

Next, we declare the grain based on the fact table that we decided. The grain will be:

* An annual sales report of property in New York City on daily basis based on different borough.

We use aggregation on the fact columns grouped by the dimension columns. The aggregation we use for the facts:

* Total Units = SUM(totalunits)
* Land Square Feet = SUM(landsqft)
* Gross Square Feet = SUM(grosssqft)
* Sale Price = SUM(price)

After getting the dimensions and the facts, we made the tables in the data warehouse. These are the tables and their properties:

|  |  |  |  |
| --- | --- | --- | --- |
| ***AddressExtension*** |  |  |  |
| **Description** | This table describes the extension of the address | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **Id** | Id of an address extension | integer | Between 1 and 999999999 |
| **Borough** | Name of a borough | varchar | Manhattan |
| **Neighborhood** | Name of a neighborhood | Varchar | ALPHABETCITY |
| **Zipcode** | Zipcode of the building | Int | 10051 |
| **Extension** | Combination of the borough, neighborhood, and zipcode | varchar | BELLEROSE, Queens, 11426 |
| **Primary Key** | id | | |
| **Foreign Keys** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***BuildingAddress*** |  |  |  |
| **Description** | This table describes the address of the building | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **Id** | Id of an address | integer | Between 1 and 999999999 |
| **address** | The address of the building | varchar | 300 EAST 2ND STREET |
| **Primary Key** | id | | |
| **Foreign Keys** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***BuildingCategory*** |  |  |  |
| **Description** | This table describes the category of the building | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **Id** | Id of a category | integer | Between 1 and 999999999 |
| **buildingcategory** | Category of a building | varchar | 07 RENTALS - WALKUP APARTMENTS |
| **Primary Key** | id | | |
| **Foreign Keys** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***yearbuilt*** |  |  |  |
| **Description** | This table describes the year of when a building was built | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **Id** | Id of a year | integer | Between 1 and 999999999 |
| **yearbuilt** | Year of which a build was built | integer | 1900 |
| **Primary Key** | id | | |
| **Foreign Keys** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***saledate*** |  |  |  |
| **Description** | This table describes the date of which the sales was done | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **Id** | Id of a date | integer | Between 1 and 999999999 |
| **year** | The year of sales | Integer | 2014 |
| **Month** | The month of sales | Integer | 10 |
| **Dayofmonth** | The day of sales in week | Integer | 22 |
| **dayofyear** | The day of sales in year | integer | 234 |
| **rawdate** | The raw data of date | date | 2012/05/02 |
| **Primary Key** | id | | |
| **Foreign Keys** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***annualsalesfact*** |  |  |  |
| **Description** | This table describes the facts of the annual property sales | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| **buildingcategoryid** | Id of a category | Integer | Between 1 and 999999999 |
| **buildingaddressid** | Id of an address | integer | Between 1 and 999999999 |
| **addressextensionid** | Id of an addressextensionid | integer | Between 1 and 999999999 |
| **yearbuiltid** | Id of a year | integer | Between 1 and 999999999 |
| **saledateid** | Id of a date | Integer | Between 1 and 999999999 |
| **totalunit** | Sum of residential unit and commercial unit | integer | 13 |
| **landsqft** | Land square feet | integer | 2400 |
| **grosssqft** | Gross square feet | integer | 3800 |
| **price** | Price of the property | integer | 800000 |
| **factcount** | Count of the fact record aggregation | integer | Between 1 and 99999999 |
| **Primary Key** | addressextensionid,buildingcategoryid, buildingaddressid, yearbuiltid,saledateid | | |
| **Foreign Keys** | Addressextensionid,buildingcategoryid, buildingaddressid, yearbuiltid,saledateid | | |

After we made the tables, we looked onto the schema diagram. And the diagram follows star schema because the tables are centered on a fact diagram.

Diagram for the fact and dimension tables:

Diagram

Description automatically generated

## 2.2 Hadoop Implementation

For Hadoop implementation, we created one fact table to contain all the records from the datasets. Based on the analytics we did from the previous section. The columns used are:

|  |
| --- |
| -BOROUGH |
| -NEIGHBORHOOD |
| -BUILDING CLASS CATEGORY |
| -ADDRESS |
| -ZIP CODE |
| -TOTAL UNITS |
| -LAND SQUARE FEET |
| -GROSS SQUARE FEET |
| -YEAR BUILT |
| -SALE PRICE |
| -SALE DATE |

The table created looks like this:

Text

Description automatically generated

Detail for fact table:

Graphical user interface

Description automatically generated with low confidence

Query for create table:

create table annualsalesfacts(

borough varchar(50),

neighborhood varchar(50),

zipcode char(5),

address varchar(50),

buildingcategory varchar(50),

yearbuilt int,

saledate date,

totalunit int,

landsqft int,

grosssqft int,

saleprice int

)

ROW FORMAT

DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

After that. We make directory in HDFS under root named ‘nysales’. And creating the input and output directories. These directories are for containing the input and output data.

Text

Description automatically generated

Inside of Input directory:

Graphical user interface

Description automatically generated

Inside of Output directory:

Text

Description automatically generated

## 2.3 Reflective analysis of using a data warehouse vs Hadoop.

Data Warehouse:

Using data warehouse for data architecture is quite time consuming as we must set the tables for the determined dimensions. And the margin of error from this process is quite large. It needs precision to determine which column is needed and which column is not needed. And due to the aggregation of data, we have to add count to the fact table. The count is to mark which record is incorporated with the fact records. For this project, the final tables created for data warehouse are 6. Consisting of 5 dimensions and 1 fact table. And from 21 columns that are available from the datasets, we are using 11 columns for the analytics. For the final design on fact table, we created 10 attributes. And we only store the ID and the numeric fact for the fact table.

Hadoop:

Creating data architecture for Hadoop system is more straightforward than data warehouse. We only need to create one fact table for the analyzed columns. This makes the error possibility is less likely than the case of relational database. As the architectural is parallel, the engine works independently and making data manipulation is easier than data warehouse. For our project, we created one fact table in Hive based on the analyzed columns. Total columns created are 11 columns. And we store the whole data from the dataset. Due to the fact that we only have one table and storing the whole dataset, we predict that the fact table will be bigger than the fact table in data warehouse.

3. Data Preparation

## 3.1 Relational Data Warehouse Implementation

## ETL considerations

## ETL Process Flow with description

General flow of the ETL:

Diagram

Description automatically generated

The ETL flow consists of 4 steps. That are DB connection and data source setting, data extraction, loading dimensions’ data, and loading the facts’ data. First, we connect the flow to the target DB by putting the hostname, database name, and the credentials. And then, we select the data we want to load to the database in the excel reader node. After selecting the data, we move on to the extraction metanode.

Inside the extraction metanode:

A picture containing diagram

Description automatically generated

We filter the row on the date column and then set it to exclude rows by attribute value and by only missing values match in the matching criteria. And then we use string to date and time node to change the type of the date column into date type. After that, we filter the column based on the used columns and unused columns from the previous insight. Then, we used Rule Engine node. This is for replacing values in borough column to the name of the borough. After that, we use Column Resorter to change the order of the column. Next, we use string manipulation to remove blank spaces for come column. Then, we use column aggregator to combine neighborhood, borough, and zipcode columns into one column. Then, we use column rename node to rename the column into ‘Extension’ column.

Next, we move on to the dimension data steps. We have 7 dimension tables. First, we look inside addressextension, address, yearbuilt, category metanode:

A picture containing timeline

Description automatically generated

We get the column of the respective nodes by the column filter. Then, we group by the column, so that only unique non repetitive values that can get into the tables. And then, we treat the missing values in missing value node. We decide to treat missing values of integer by putting the most frequent value. For string data type, we decide to put fix value ‘UNKNOWN’. After that, we rename the column according to the column name in the table using column rename node. After that, the DB query reader node reads the target table to be referenced in the flow. After the flow read the table, the empty table switch read the data. If there is nothing readable from the reader, the switch directed the flow immediately to the top reference row filter. Reference row filter node is for matching the data table column and reference column. After that, we load the data to the target table using DB Writer node. If the empty table switch node reads some data from DB query reader node, that makes the target table is not empty. Thus, running the bottom flow. We use joiner node to get distinct values of the data. The flow joins the target table and the data. Then, we get the distinct values by getting the left unmatched rows. And then we only load the distinct values to the target table.

After that, we look inside the metanode of saledate:

A picture containing chart

Description automatically generated

The flow is generally the same as the other dimensions. The only difference is that there is an extract date and time fields node. That is to get the year, month, and day of the date. We also treat the missing values of date type by putting the previous value of the date.

Finally, we move on to the facts loading step. Here is the inside of the facts metanode:

Diagram

Description automatically generated

In this metanode, we try to join the extracted data and the data from the dimension tables that were loaded previously so that we can get the id from each of the columns. We use inner join matching rows for the values. And then we select the id column only. Next, we convert the price column to string and convert it back to integer number. And if there’s missing values, we put the maximum number of prices to the missing value. This process is due to the fact that there’s some row that the number is too big. That it changes the data type of the price column from integer to long type. And we can’t sum the price because of that. Making the loaded price to become missing. Finally, we group by the data by all of the collected id’s, and then we sum the total units, land square feet, gross square feet, and the price. Rename the column according to the target table’s column. And finally load the facts to the facts table.

## 3.2 Hadoop Implementation

For Hadoop ETL process, we first need to extract all of the files and save it in one folder. Here, we save it in C:\Users\USER-PC\documents\nysales:

Text

Description automatically generated

We also created mapper csv file to convert the borough code into borough name. inside of the mapper:

Graphical user interface, application, table, Excel

Description automatically generated

After that, we move the files into temporary folder in the container using docker cp command. Then, we access the container and check if the files are already there. The result is the following:

Graphical user interface, text

Description automatically generated

After making sure that the files are already in the tmp folder. We make folder in HDFS using -mkdir command. Here. The folder created is nysales. And then, we create the input folder inside it. Next, we put the files from the tmp folder into the created folder in HDFS using -put command. Then, we check if the files are correctly put into the folder. The result is following:

Graphical user interface

Description automatically generated with medium confidence

ETL using Apache Pig

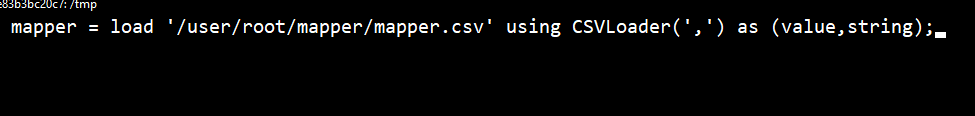
For this process, we chose to use the Apache Pig. First, we access the pig using pig command. And after that, we load the CSV loader to the pig using this command:

DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader();

After that, we load all the csv files to the pig. And then we declare all the columns that the loader can read. We name the data ‘nysales’. The command used is following:

nysales = load'/user/root/nysales/input/\*.csv' using CSVLoader(',') AS (borough:chararray, neighborhood:chararray, buildingcategory:chararray, tcp:chararray, bl:chararray, lt:chararray, em:chararray, bcp:chararray, address:chararray, an:chararray, zipcode:chararray, ru:chararray, cu:chararray, totalunits:chararray, landsqft:chararray, grosssqft:chararray, yearbuilt:chararray, ts:chararray, bs:chararray, saleprice:chararray, saledate:chararray);

Next, we load the mapper csv and declare the column. The command:



mapper = load '/user/root/mapper/mapper.csv' using CSVLoader(',') as (value, boroughname);

Next, we join the mapper and the nysales by the borough code and the mapper value. The command used:

Text

Description automatically generated

joined = join nysales by borough, mapper by value;

After joining both data, we generate data by taking some of the columns based by the hive table that we created before. The command used:

nysalesjoined = foreach joined generate boroughname, neighborhood, zipcode, address, buildingcategory, yearbuilt, saledate, totalunits, landsqft, grosssqft, saleprice;

Next, we filter the rows that have missing values by simply removing them. The command used:

nysalesjoined = FILTER nysalesjoined BY boroughname != '';

nysalesjoined = FILTER nysalesjoined BY neighborhood != '';

nysalesjoined = FILTER nysalesjoined BY zipcode != '';

nysalesjoined = FILTER nysalesjoined BY address != '';

nysalesjoined = FILTER nysalesjoined BY buildingcategory != '';

nysalesjoined = FILTER nysalesjoined BY yearbuilt != '';

nysalesjoined = FILTER nysalesjoined BY saledate != '';

nysalesjoined = FILTER nysalesjoined BY totalunits != '';

nysalesjoined = FILTER nysalesjoined BY landsqft != '';

nysalesjoined = FILTER nysalesjoined BY grosssqft != '';

nysalesjoined = FILTER nysalesjoined BY saleprice != '';

Next, we remove the comma symbol from the columns. Because the original datasets contain some comma in the value. This symbol can be mistaken as separator and distorted the structure of the data. The command used:

nysalesjoined = foreach nysalesjoined generate boroughname, neighborhood, zipcode, REPLACE(address, ',','') as address, buildingcategory, yearbuilt, saledate, REPLACE(totalunits,',','') as totalunits, REPLACE(landsqft,',','') as landsqft, REPLACE(grosssqft,',','') as grosssqft, REPLACE(saleprice,',','') as saleprice;

After that, we remove the dollar sign from the saleprice column. Because the dataset contains dollar symbol. And because we declare the saleprice column as INT. This issue can be a problem because there will be a data type error. Thus, we have to remove the symbol. The command used:

nysalesjoined = foreach nysalesjoined generate boroughname, neighborhood, zipcode, address, buildingcategory, yearbuilt, saledate, totalunits, landsqft, grosssqft, REPLACE(saleprice,'\\\$','') as saleprice;

Next, we cast some of the columns into the data type according to the table in the hive. The command used:

nysalesjoined = foreach nysalesjoined generate boroughname, neighborhood, zipcode, address, buildingcategory, (INT)yearbuilt, ToDate(saledate, ‘MM/dd/yyyy’) as (saledate:date), (INT)totalunits, (INT)landsqft, (INT)grosssqft, (INT)saleprice;

After that, we cast saledate back to char array and format the date to yyyy-MM-dd. Because this format is the default date format of hive. The command used:

nysalesjoined = foreach nysalesjoined generate boroughname, neighborhood, zipcode, address, buildingcategory, yearbuilt, ToString(saledate, ‘yyyy-MM-dd’) as (saledate:chararray), totalunits, landsqft, grosssqft, saleprice;

Finally, we store the data we have transformed to the HDFS folder and create the output folder in it. We also give the comma separator to the file. The command used:

STORE nysalesjoined INTO '/user/root/nysales/output' USING PigStorage (',');

Next, we exit the pig using EXIT command. And then we check the file created to the folder. The result is following:

Graphical user interface

Description automatically generated with low confidence

Next, we copy the file to the container tmp folder. The command used:

hadoop fs -get /user/root/nysales/output/part-m-00000 /tmp/nyannualsalesfacts.csv

Inside of the csv file generated:

A picture containing text

Description automatically generated

After we make sure that the csv is correct, we execute the hive again. And then we load the csv file into the hive table. The command used:

load data local inpath '/tmp/nyannualsalesfacts.csv' overwrite into table annualsalesfacts;

we check the count if the data is successfully loaded. The result is following:

Graphical user interface, text

Description automatically generated

## 3.3 Reflective analysis of data preparation in relational data warehouse vs Hadoop.

Data Warehouse:

Preparing ETL in data warehouse is quite difficult and time consuming if one doesn’t understand the flow of the ETL. But once set and the flow is working properly, the application is more practical than the Hadoop system. Data warehouse is needed to be set properly because it can only analyze processed and structured data. If the flow is not running properly, the job wouldn’t be done correctly. And we think that processing data in data warehouse is less agile because the configuration is quite fixed. And it’s quite difficult to change the flow. Data warehouse has been around for long time. So, the stability of data warehouse is reliable. For our project, we separate the entire process into 4 sections. First, we connect the flow to the database and the input data. Next, we extract the required columns from the dataset. Next, we load the dimensions into the database. Lastly, we load the facts to the database. The problems we had when we designed the flow is when we handle missing value, determining the right node to be used, and joining the right column.

Hadoop:

The Hadoop system ETL is easier than the data warehouse and more straightforward. But the application is kind of difficult as we have to repeat the same process from scratch over and over. Using Hadoop for processing data is straightforward because Hadoop can process any kind of data like raw, structured, unstructured, and semi-structured data. And the agility of Hadoop is highly agile because the flexibility of Hadoop in configuration. And Hadoop can work well with large volume of data and with variety of data types. For our project, at first, it was confusing when using HDFS and its frameworks because it is our first time to use it. But after we dive more into it, configuring ETL process in Hadoop is relatively easy. The problems we had when we configured the process in Hadoop is that some of the data is not cleaned. Like for unit price column, the column has dollar symbol. It needs to be removed as the hive won’t be able to store it. And some of the columns have comma symbol. This is a problem because it can distort the structural integrity of the data. Because we are using CSV data. And we also had a problem to decide which command is right for which action as we don’t have any experience with linux.

4. Reporting System

## 4.1 Relational Data Warehouse Implementation

A picture containing text, map, indoor

Description automatically generated

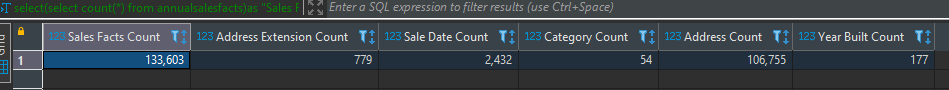
For the reporting flow, we read the data from facts table. After that, we join the dimension and the fact tables. Inner join them by the ids. And then get the values of the table. Then, we use the data to report node to send the final data to reporting editor.

We made some table viewer on the reporting to check whether the data is loaded correctly. The result is the following:

Table

Description automatically generated

We also check the count for each of the table of dimension and fact. Here, we can see that the fact table has 133,603 rows. The extension dimension has 779 rows. The date dimension has 2432 rows. The building category count has 54 rows. The address count has 106,755 rows. And the yearbuilt count has 177 rows.



We also try to know the size of each table after the data loading. We want to estimate the size increasing if more data is loaded to the table. The total combined size of fact and dimension tables is 28MB. The fact and the address tables are the highest in size. This is because both tables contain distinct value for each row.

A screenshot of a computer

Description automatically generated

We also want to know the total records we got based on the borough. We count the rows for each borough. Note that this count is after aggregation of records. So, there would be no duplication for each dimension IDs. We can see that there are 5,029 rows for Bronx, 32,755 for Brooklyn, 36,431 for Manhattan, 40,639 for Queens, 18,749 for Staten Island.

Chart, bar chart

Description automatically generated

## 4.2 Hadoop Implementation

We use knime for reporting of the fact table that we have created in hive. We simply use the hive connection node. And then we read all of the data that we have loaded in hive. And then we send the data to reporting editor.

Chart

Description automatically generated

We made some table viewer on the reporting to check whether the data is loaded correctly to the hive table. We got 147,865 rows from it. The result is the following:

Table

Description automatically generated

We also try to check the size of the fact table in hive. The size is 23Mb. This is 7Mb larger than the fact table in postgre. This makes sense because the data here is raw data from the dataset that we didn’t aggregate the data with the same dimension IDs.

Text

Description automatically generated

We also count the records based on borough for the data in hive. We got 5,243 from Bronx. 34,243 from Brooklyn, 46,024 from Manhattan, 42,778 from Queens, 19,557 from Staten Island.

The difference from the fact table that we got from postgre is 14,262 rows. Again, this is because we didn’t aggregate data for the hive table.

Chart, bar chart

Description automatically generated

## 4.3 Reflective analysis of result in relational data warehouse vs Hadoop.

Data Warehouse:

Steps of reporting in data warehouse is relatively easy to implement. But the nodes needed for the reporting flow is more than hadoop. Because the joins that are needed to get the data of the facts table are so many. By looking at the reporting flow, the flow looks intricate. And looking at the total of space needed for the dimension and the facts. They need larger space than Hadoop. The total of the space needed for data warehouse is 28MB. But the fact table requires less space because the data is aggregated. Queens got the most records after the aggregation.

Hadoop:

The reporting system for the hadoop is more straight forward than the data warehouse. The nodes required for the flow are fewer than data warehouse. Because there’s no join required because the data is already a fact table. Overall flow of reporting is much simpler too. It only needs to read the data and then send it to report system. We found that the total size of the records is 23 MB. 5 MB less than we did in data warehouse. And the total record is mostly come from Manhattan before the aggregation. Nevertheless, both systems are relatively easy to report.

Comparison:

Total Table:

Hadoop: 1 Table

Data Warehouse: 6 Tables

Total data input:

Hadoop: 147,865 Rows

Data Warehouse: 133,603 Rows

Total Size:

Hadoop: 23Mb

Data Warehouse: 28Mb

Conclusions

In conclusion, both of the systems have their own advantages and disadvantages. The advantages and disadvantages based on this project that we found are:

Data Warehouse:

Using data warehouse for this project is quite tricky at the beginning. We have to precisely design the architecture of the data so that the data is following the schema. And the data used is structured and processed data. But then, once the preparation is already set up, it is relatively easy to maintain daily load of data. The space required for total data is larger than Hadoop. But the total rows are already aggregated.

Hadoop:

For us, Hadoop is a powerful system that can work similarly to data warehouse. Hadoop is easy to implement and easy to understand because of the straightforwardness. Manipulating data in Hadoop is convenient because we can do it directly. Hadoop can also process unstructured and raw data. It also works well with large volume of data. But to continuously loading data to the system, it’s quite difficult because we have to set it up over and over again. The space required is less than data warehouse. But the data is not aggregated. At the end of the day, Hadoop will not replace data warehouse as they both have their advantages and disadvantages.