Analysis of US 2020 Presidential Election Contributions Using a Data Warehouse and Hadoop

Sean Onyskiw

Table of Contents

Analysis of US 2020 Presidential Election Contributions Using a Data Warehouse and Hadoop

Sean Onyskiw

INTRODUCTION

The Federal Elections Commission maintains detailed records of contributions made to political candidates and organizations. This data includes multiple details for each contribution, including name, address, amount, employer, job title, and the organization or candidate receiving the donations. The data also contains records of any revisions to the amount donated for each contribution.

Utilizing this data can provide insights into elections. The report will try to identify any trends found in the contribution data. This report describes the process for building a data warehouse to store contribution data. The report also describes the process of building reports for the data warehouse using KNIME, and the results of the analysis of the data. A second solution for storing data using Hadoop will also be constructed and compared to the data warehouse solution.

PURPOSE

This document defines the structure of the data warehouse and the reports generated from it. The document provides the results of the analysis of the reports and conclusions that can be drawn from the data. The report also includes an alternative method using Hadoop to store the data.

PROJECT SUMMARY

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

A. Objectives

The objective of this document is to establish a data warehouse and the reporting process used for the warehouse. Hadoop was then used as an alternative implementation and compared to the data warehouse solution.

B. Scope

The scope of the project will be to analysis the default data for DAAN 825. This data includes contributions from four states, California, Missouri, New York, and Texas. The data was collected from July of 2019 to August of 2020.

C. References

Data was taken from the reference data used for DAAN 825 which is sourced from the Federal Election Commission.

D. Outstanding Issues

Only a limited analysis is presented in this document. Further analysis using different variables such as zip code and comparing individual vs organization contributions can potentially reveal greater insights into campaign contribution trends.

REQUIREMENTS DEFINITION

A. Goals

The goal of this document is to define the specifications of a data warehouse and a Hadoop system for storing contributions from the 2020 election and performing an analysis of the data.

B. Usability Requirements

The system must ensure ease of use when generating reports so that users unfamiliar with the system can easily retrieve needed information.

C. System Security Requirements

Currently this system has no security beyond a simple password to access the databases. If the system where to be used in a production environment, access control would need to be established. Writing and removing data from the system would need to be restricted to only a select group, or automated in a way that prevents data loss or corruption. Read only accounts would likely be established for users only needed to perform an analysis of the data.

D. Business Questions

The analysis of this document will look at trends in contributions leading up to the election. Specifically, the following questions will be answered:

- When were contributions made? Were contributions steady or did they mostly occur at a certain time?
- Who raised the most money in the primary? Who raised the most in the general election? Did the person who raised the most win?
- What are some of the top occupations for donating money?

E. Data Requirements

Relevant data will be stored in a data warehouse using a star schema. A second storage system will be created using Hadoop and compared to the data warehouse.

F. Design Constraints

Data may not be consistent or have errors. Correction of data may be required before analysis of the data can be completed.

CONSIDERATIONS

One of the first considerations for the structure is what data will be used in the data warehouse. The available data is shown in the table below, however not all data is relevant or unique. Some data will not be included in the warehouse, which is describe below the table.

Available Data:

Name	Description	Example
filing_id	Filing ID	SA17A
linenumber	Line Number	SA17A
flag_orgind	Indicates individual or organization contribution	IND
org_name	Organization name if applicable	Citibank
last_name	Individual last name	Aberle
first_name	Individual first name	Elizabeth
middle_name	Individual middle name, can be just an initial	W.
prefix	Individual name prefix	Dr.
suffix	Individual name suffix	Jr.
address_one	Contributor address line one	635 Calle Arroyo
address_two	Contributor address line two	Apt 102
city	Contributor city	Thousand Oaks
state	Contributor state	CA
zip	Contributor zip code	91360
employer	Contributor employer	None
occupation	Contributor occupation	Student
amount	Contribution amount	5
date	Contribution date	2020-01-07
aggregate_amount	Aggregate contribution amount	459
memo_code	Memo code	25
memo_text	Memo text	* Earmarked Contribution: See Below
tran_id	Transaction ID	5461214
back_ref_tran_id	Additional part pf transaction ID	5461214E
back_ref_sched_name	Scheduled Name	SA17A

prigen	Primary or general election and year	P2020
cycle	Year of election	2020
fecid	FEC ID	C00693234
committee_name	Name of organization recieving contribution	"Warren For President, Inc."

The usefulness of each attribute was also evaluated:

Description	Notes	Copied to warehouse
filing_id	Only small number of unique codes are used so most entries share the same codes. As a result, there is not much insight to gain.	N
linenumber	Same as Filing ID	N
flag_orgind	Usefull to identify individual or organization	Y
org_name	Not kept as data will be viewed in aggregate	N
last_name	Not kept as data will be viewed in aggregate	N
first_name	Not kept as data will be viewed in aggregate	N
middle_name	Not kept as data will be viewed in aggregate	N
prefix	Not kept as data will be viewed in aggregate	N
suffix	Not kept as data will be viewed in aggregate	N
address_one	Not kept as data will be viewed in aggregate	N
address_two	Not kept as data will be viewed in aggregate	N
city	Will be kept to identify contribution locations. Zip code can also potentially be used to identify city.	Y
state	Useful to easily identify state	Y
zip	Useful if a more specific location is required.	Y
employer	Kept to identify if employees of certain companies donate larger amounts.	Y
occupation	Kept to see the amounts each occupation donates.	Y
amount	Amount of contribution. Required to complete analysis of donation amounts.	Y
date	Useful for analysis to determine when contributions are made.	Y
aggregate_amount	Not kept as the analysis will be looking at an aggregate of each contribution.	N
memo_code	Not useful for analysis.	N
memo_text	Not useful for analysis.	N
tran_id	Identifies each unique transaction. Can be used as a database key.	Y

back_ref_tran_id	Not useful for analysis.	N
back_ref_sched_name	Not useful for analysis.	N
prigen	Needed to identify whether donation is for primary or general election	Y
cycle	Not needed since only the 2020 election will be analyzed.	N
fecid	Similar to the filing ID, only a small number of unique codes are used.	N
committee_name	Identifies the organization that recieves the contribution.	Y

After the utility of each attribute was considered, only 8 of the 28 attributes were kept to be stored in the data warehouse.

DOCUMENT CHANGE LOG

Change Date	Version	CR #	Change Description	Author and Organization
04/07/2024	1.0		Initial creation.	Sean Onyskiw
04/26/2024	2.0		Added Hadoop Implementation and revised data warehouse implementation.	Sean Onyskiw

2. ARCHITECTURE DESIGN

2.1 Relational Data Warehouse

Data Dictionary

As described in the considerations section above, only some data will be kept in the warehouse. The variables that will be stored are shown below:

Name	Description	Type	Values
flag_orgind	Indicates individual or organization contribution	string	IND, ORG
state	Contributor state	string	CA, NY, MO, TX
zip	Contributor zip code	int	12345
amount	Contribution amount	double	100.00
date	Contribution date	date	2020-01-01
tran_id	Transaction ID	string	5461214, A048AC853E89049C4A6A
prigen	Primary or general election and year	string	P2020, G2020
committee_name	Identifies the organization that receives the contribution.	string	Biden For President, Donald J. Trump For President, Inc.
employer	Contributor employer	string	Stanford University
city	Contributor city	string	Los Angeles
occupation	Contributor occupation	string	Teacher

Tables schemas

The schema and table structures are shown below. The database is arranged in a star schema, with the facts table as the center of the star and the committee name, origin, primary or general, state, transaction ID, date, and zipcode tables as dimension tables. The primary key of the facts table is a combination of all of the dimension IDs, as well as the amount. This results in all columns as part of the primary key. This was needed to avoid deletion of data as some records had the same values for all attributes, only differing in the amounts.

There are 2,876,705 records in the dataset. When a duplication check was performed using all dimensions and excluding the amount, there were 525,698. Even including first and last names in resulted in 288,915 duplicate rows. When the amount was included there were still 168,701 duplicated rows, likely indicating that some individuals made multiple donations on the same day, and for the same amount. When transaction ID was added, the number of duplicate rows was reduced to 7, indicating that transaction ID is needed to identify unique records.

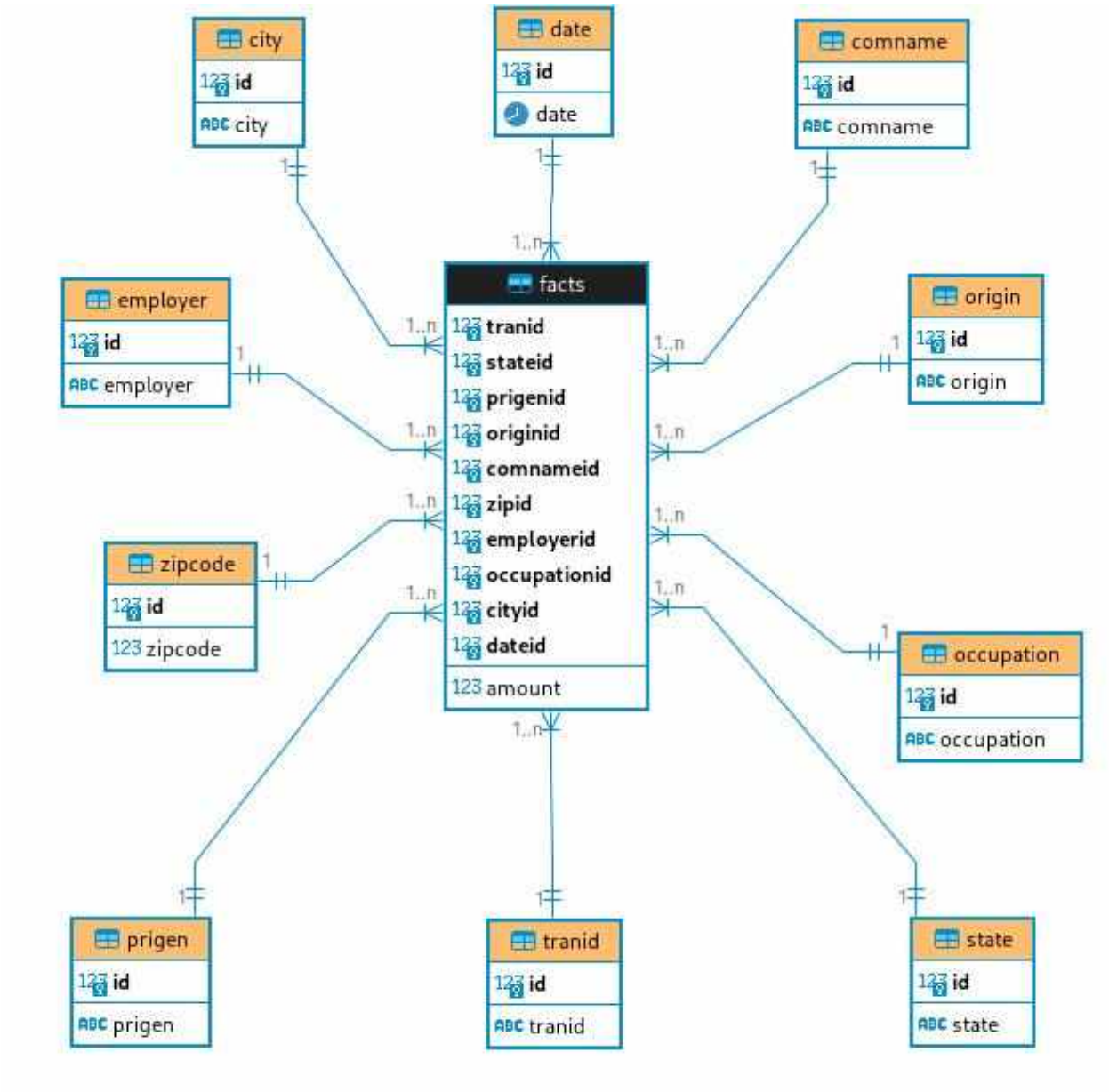


Diagram of Data Warehouse Schema

Table structures of the schema are provided below:

comname			
Description	This table contains the committee names.		
Attribute	**Description**	**Type**	***Examples of values***
id	Committee ID	Serial Integer	Between 1 and 999999999
comname	Committee Name	String	Biden For President Donald J. Trump For President, Inc.
Primary Key	id		
Foreign Keys	None		

origin			
Description	This table contains the origin of the contribution.		
Attribute	**Description**	**Type**	***Examples of values***
id	Origin ID	Serial Integer	Between 1 and 999999999
origin	Origin of contribution	String	IND, ORG
Primary Key	id		
Foreign Keys	None		

prigen			
Description	This table whether the contribution was for the primary or general election.		
Attribute	**Description**	**Type**	***Examples of values***
id	Election ID	Serial Integer	Between 1 and 999999999
prigen	Primary or General	String	P2020, G2020
Primary Key	id		
Foreign Keys	None		
state			
Description	This table contains the state the contribution originated from.		
Attribute	**Description**	**Type**	***Examples of values***

id	State ID	Serial Integer	Between 1 and 999999999
state	Abbreviation of state	String	CA, NY, MO, TX
Primary Key	id		
Foreign Keys	None		

tranid			
Description	This table contains the transaction ID converted to a serial integer from the original id.		
Attribute	**Description**	**Type**	***Examples of values***
id	Transaction ID	Serial Integer	Between 1 and 999999999
tranid	Original transaction ID	String	5461214, A048AC853E89049C4A6A
Primary Key	id		
Foreign Keys	None		

employer			
Description	This table contains the employer ID converted to a serial integer from the original id.		
Attribute	**Description**	**Type**	***Examples of values***
id	Employer ID	Serial Integer	Between 1 and 999999999
employer	Employer Name	String	Stanford University
Primary Key	id		
Foreign Keys	None		

occupation			
Description	This table contains the occupation ID converted to a serial integer from the original id.		
Attribute	**Description**	**Type**	***Examples of values***
id	Occupation ID	Serial Integer	Between 1 and 999999999
tranid	Ocupation Name	String	Teacher
Primary Key	id		
Foreign Keys	None		

city			
Description	This table contains the city ID converted to a serial integer from the original id.		
Attribute	**Description**	**Type**	***Examples of values***
id	City ID	Serial Integer	Between 1 and 999999999
tranid	City Name	String	Los Angeles
Primary Key	id		
Foreign Keys	None		

facts			
Description	The facts table contains the measurements of each transaction.		
Attribute	**Description**	**Type**	***Examples of values***
tranid	Id of the transaction	Integer	Between 1 and 999999999
comnameid	Name of the committee	Integer	Between 1 and 999999999
zipid	Zip code	Integer	Between 1 and 999999999
stateid	State ID	Integer	Between 1 and 999999999
prigenid	Primary or General ID	Integer	Between 1 and 999999999
originid	Origin ID	Integer	Between 1 and 999999999
amount	Contribution amount	Double	Between 0.00 and 999999999.99
dateid	Date of Contribution	Integer	Between 1 and 999999999
employerid	Employer ID	Integer	Between 1 and 999999999
occupationid	Occupation ID	Integer	Between 1 and 999999999
cityid	City ID	Integer	Between 1 and 999999999
Primary Key	(tranid, comnameid, stateid, prigenid, originid, zipid. dateid, emplyerid, occupationid. cityid)		
Foreign Keys	tranid, comnameid, stateid, prigenid, originid, zipid. dateid, emplyerid, occupationid. cityid		

2.2 Hadoop Implementation

Since the data is not split into dimension tables, the architecture will be a simpler schema with all data stored into one table. Each attribute will be stored as text.

2.3 Reflective analysis of using a data warehouse vs Hadoop.

Since there is no dimensional architecture defined, the process of getting the information into Hive is much simpler as the architecture does not need to be created before data preparation. The drawback of using this method is that the data is not normalized and there will be a significant amount of repetition in the table, especially for attributes that have few unique values.

3. Data Preparation

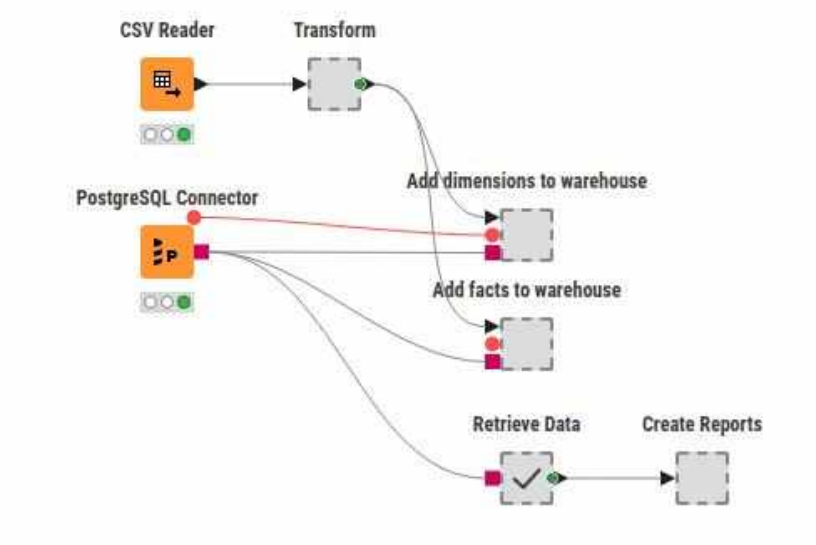
3.1 Relational Data Warehouse Implementation

ETL considerations

KNIME was used for ETL. The ETL is divided into four main parts: Storing the dimensions, storing the facts, retrieving the facts and dimensions, and generating reports. Each of these processes is described below.

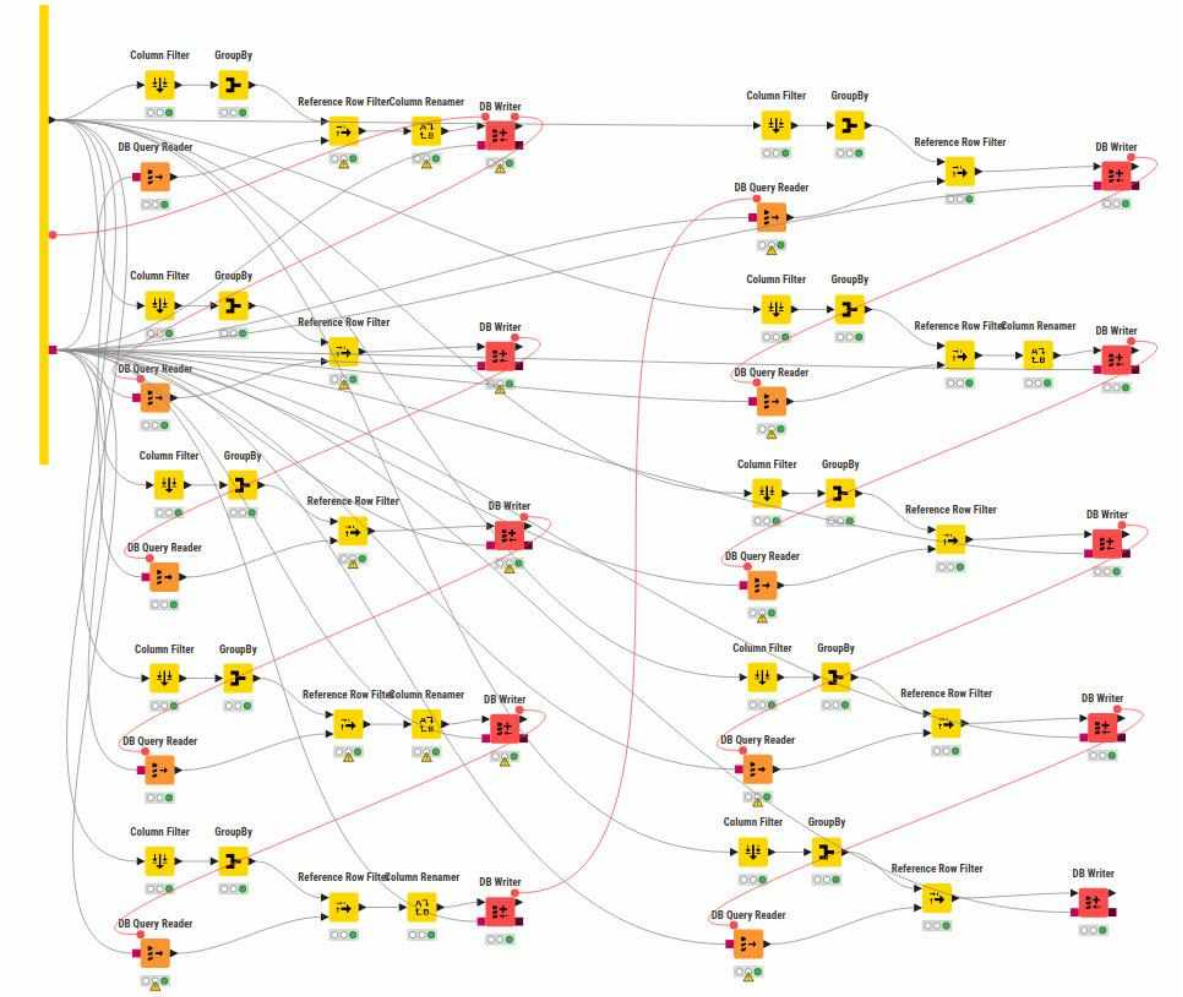
ETL Process Flow with description

The overall process flow is shown in the diagram below. New data is read from a CSV and copied into a postgreqsl database. Data is read from the database to be used for analysis.



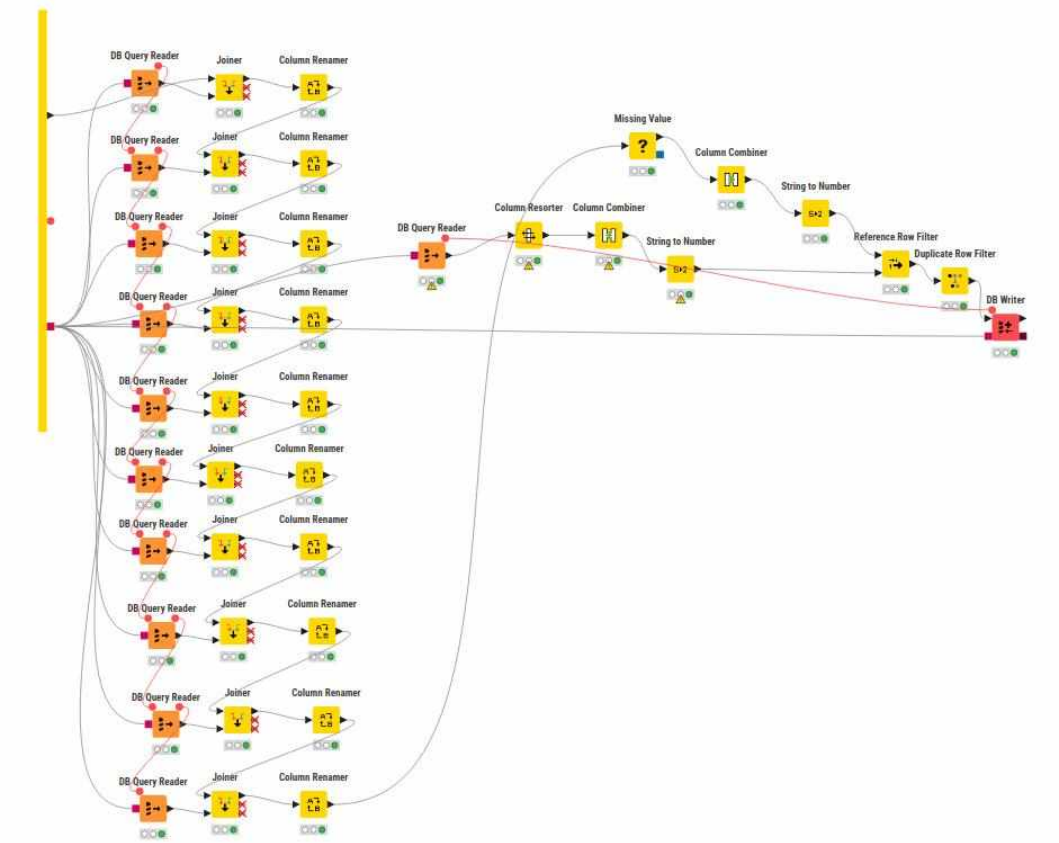
Storing the dimensions:
The CSV file is read, and unneeded columns are removed from the data. Missing zip data is replaced with a zero since it is needed for the primary key of the facts table. The dimension storing consists of several steps. Each dimension, such as state or origin first goes through a group by step to isolate unique values. In the diagram below, this step is completed by the column filter and groupby nones. The column filter reduces the table to the specified dimension and the groupby node reduces the table to only one occurrence of each dimension value.

The values are then compared to already existing values in the dimension tables. The reference row filter will remove any existing values from the groupby list. If the value does not exist in the database, it is written to the database in the DB writer step. A new row is added to that dimension table, with the new ID incremented by one from the last entry.

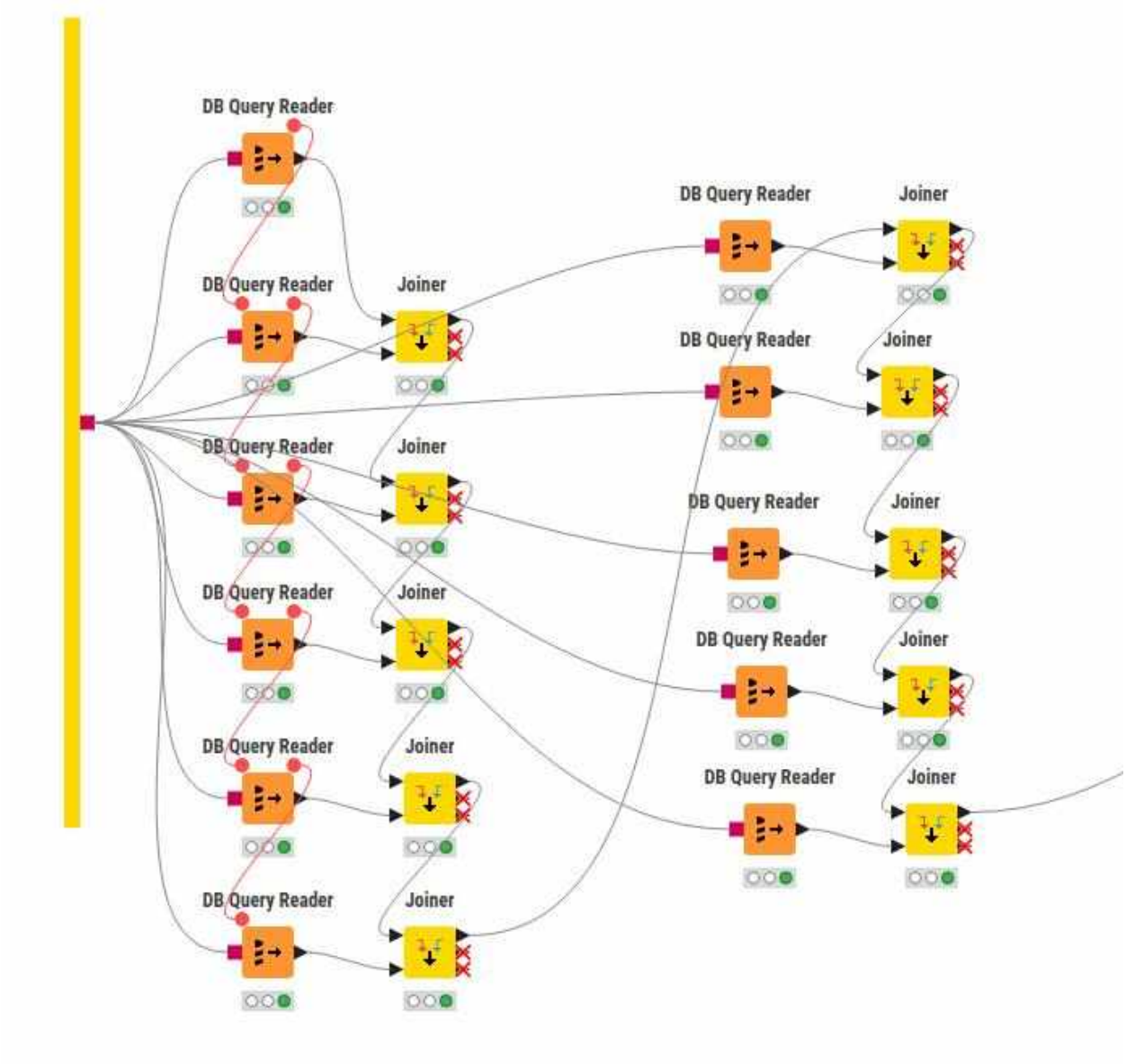


Storing the facts:
The dimension attributes of each entry into the facts table are first converted into the dimension IDs. Each dimension table is read from the database. The value of each dimension in a new entry is compared to the dimension table to find the corresponding ID number and the existing value is replaced with the id number. Missing zip codes are replaced by a 0, and a check is performed to make sure no combination of the dimension IDs are identical with an existing entry as those values from the primary key of the facts table. Duplicate rows are excluded before data is written to the database. A single string combining all the dimension IDs is created and then converted to a number. If this number exists in another record, the record is deleted before the data is sent to the database to be stored.

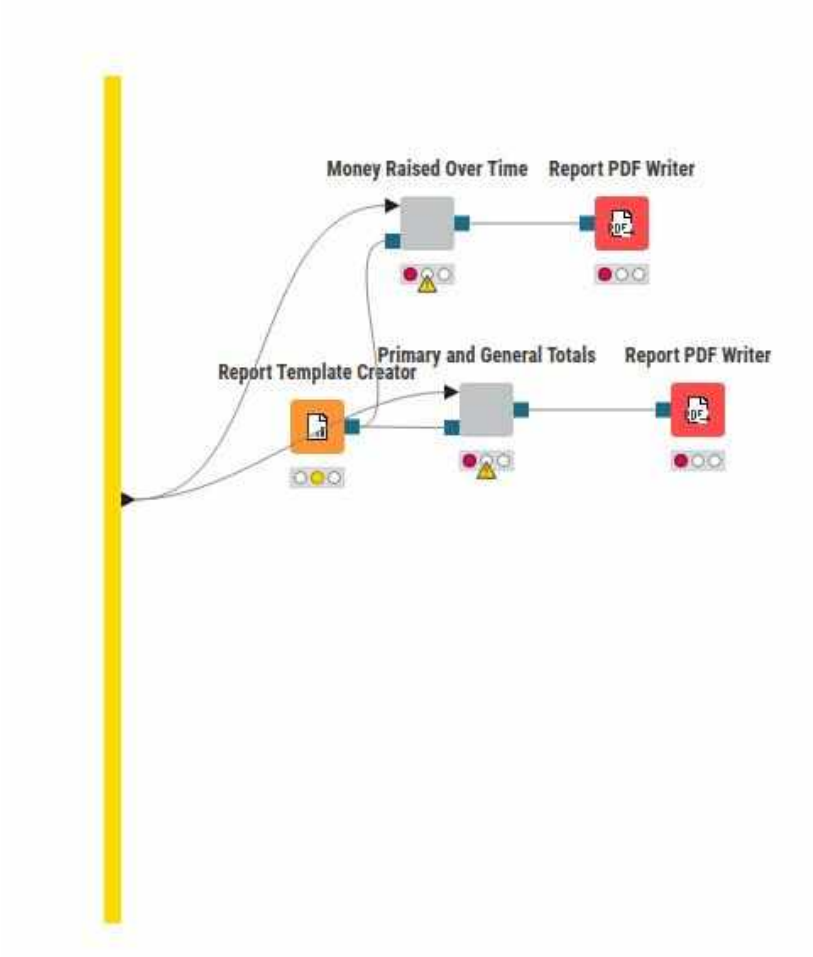
In the diagram below, the joiner node is used to convert the dimension into the dimension ID. The column is then renamed to match the column name in the database. Any duplicate rows are then removed. The final step is to check if an entry is already in the facts table. To do this, the dimension IDs are concatenated together. The concatenated value is compared to the concatenated values already existing in the database, and if a match is found, the new entry is excluded from the data written to the database.



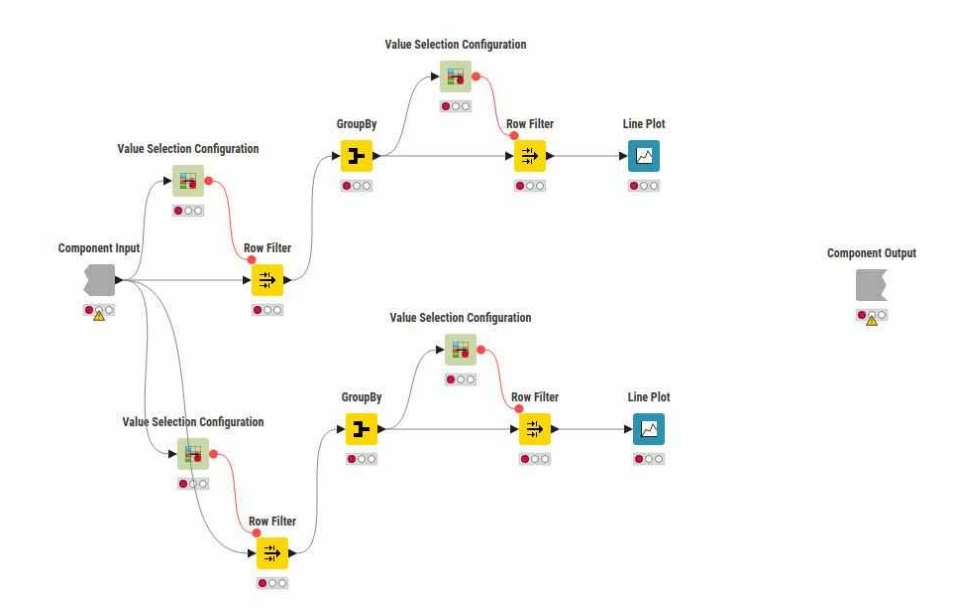
Retrieving of the facts and dimensions:
The facts table is read from the database, and the dimensions values are pulled from the dimensions tables. The dimension values are recombined with the fact table using the dimension ID. The dimension ID is then discarded.



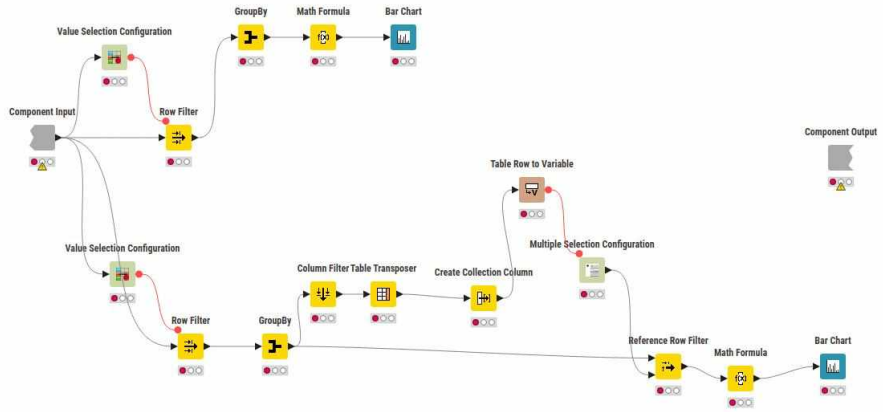
Generating Reports:
Reports are generated by filtering data to answer specific questions. The output is formatted into a report that can be printed as a PDF file. An overview of the process flow is shown in the diagram below.



Overview of report generation



Line plots of money raise over time



Bar charts to compare money raised by candidates.

3.2 Hadoop Implementation

Following construction of the data warehouse, the data transformation and storage was reconstructed using Hadoop. The same CSV files containing the data were used for the Hadoop implementation as well.

The CSV files were first copied to the Hadoop Distributed File System. Apache Pig was then used to extract and transform the data into the needed format to be then loaded into Apache

Hive. The Pig script contained only a few lines of needed code to transform the data into the same format that was initially done using KNIME.

Apache Pig:

First, the data was loading using the CSV Loader:

```
--Register CSV Loader
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader();
```

The data was then loaded from the CSV files. All data was treated as text in the schema:

```
--Load the data
data = LOAD '/project/data/*.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage() AS (filing_id:chararray,
linenumber:chararray, flag_orgind:chararray, org_name:chararray, last_name:chararray,
first_name:chararray, middle_name:chararray, prefix:chararray, suffix:chararray,
address_one:chararray, address_two:chararray, city:chararray, state:chararray,
zip:chararray, employer:chararray, occupation:chararray, amount:chararray,
tdate:chararray, aggregate_amount:chararray, memo_code:chararray,
memo_text:chararray, tran_id:chararray, back_ref_tran_id:chararray,
back_ref_sched_name:chararray, prigen:chararray, cycle:chararray, fecid:chararray,
committee_name:chararray);
```

Since each CVS file contained a header, a filter was used to remove lines with filing_id, removing the header rows:

```
--Remove header lines
data = FILTER data BY filing_id != 'filing_id';
```

Unneeded columns are removed. The remaining columns are the same columns that were kept using KNIME earlier when processing the data - the committee name, origin, primary or general, state, transaction ID, zip code, transaction date, and amount. The zip codes were casted to integer, and the amounts casted to doubles. In Pig, if any entry cannot be converted to an integer for zip code and double for amount, the existing value will be replaced by a null. These null values are addressed below.

```
--Keep needed columns and cast zip and amount to numeric
data = FOREACH data GENERATE flag_orgind, state, (int)zip, (double)amount, tdate,
tran_id, prigen, committee_name, city, employer, occupation;
```

Records that contained null values in the amount column were removed from the data set as those records have no value for the data analysis.

```
--Eliminate null amounts
data = FILTER data BY amount is not null;
```

The zip code and amount were changed back from numeric to string.

```
--Change data back to chararray
data = FOREACH data GENERATE flag_orgind, state, (chararray)zip, (chararray)amount,
tdate, tran_id, prigen, committee_name, city, employer, occupation;
```

Since the zip code column is now a string again, any null values can be replaced with a ‘0’.

```
--change nulls in zip to 0
data = FOREACH data GENERATE flag_orgind, state, (zip is not null ? zip : '0') as zip,
amount, tdate, tran_id, prigen, committee_name, city, employer, occupation;
```

Duplicate rows are removed:

```
data = DISTINCT data;
```

The processed data is then outputted to HDFS.

```
STORE data INTO '/project/output' USING PigStorage ('|');
```

Apache Hive

The processed data from Pig was then loaded into Hive. The output from Pig was first converted into a CSV file on the local filesystem:

```
hadoop fs -getmerge /project/output/ /root/project/hive/processed.csv
```

The database and table were then created in Hive. Similar to the Pig script, the table uses text for all attributes in the table.

```
CREATE DATABASE IF NOT EXISTS projectdb;

CREATE TABLE projectdb.contribfacts(
      flag_orgind VARCHAR(50),
      state VARCHAR(2),
      zip VARCHAR(5),
      amount VARCHAR(20),
      tdate VARCHAR(12),
      tran_id VARCHAR(50),
      prigen VARCHAR(10),
      committee_name VARCHAR(50),
city VARCHAR(50),
employer VARCHAR(50),
occupation VARCHAR(50)
)
ROW FORMAT
DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
```

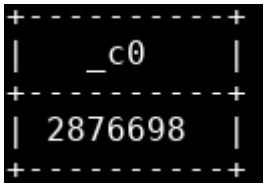
The data was then loaded into Hive:

```
load data local inpath '/root/project/hive/processed.csv' overwrite into table
projectdb.contribfacts;
```

The data loading can be verified using:

```
select count(*) from projectdb.contribfacts;
```

This returns a count of the records.



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

Preparing the data was significantly simpler using Apache Pig and Hive versus using the data warehouse and KNIME. KNIME involved a much more complicated system that involved converting attributes into IDs, storing the IDs and attributes in separate tables, and verifying that no duplicate records existed in the dimension tables. The dimension IDs then needed to be combined into a key for the facts to confirm no duplicates are written to the facts table.

When retrieving the data, a join needed to be performed for each attribute stored in a dimension table. In the data warehouse for this project, that included every attribute except for amount. This storing and retrieving of attributes took a significant time to create in KNIME and takes significant processing time to implement.

By contrast, the implementation of Hadoop and storing the data in Hive was much simpler. Data was left as text, and no splitting of the data into separate tables was performed. Deleting duplicate data was as simple as using the DISTINCT function in Pig. Loading the data into Hive required a few simple commands, and all data can be loaded into KNIME using just one node for the Hive query. The primary drawback of the Hadoop implementation is that none of the redundancy of attributes is reduced. The tradeoff between the two methods is that the data warehouse requires more time to implement and more processing time when retrieving and storing data, but had the advantage of reduction of storage size.

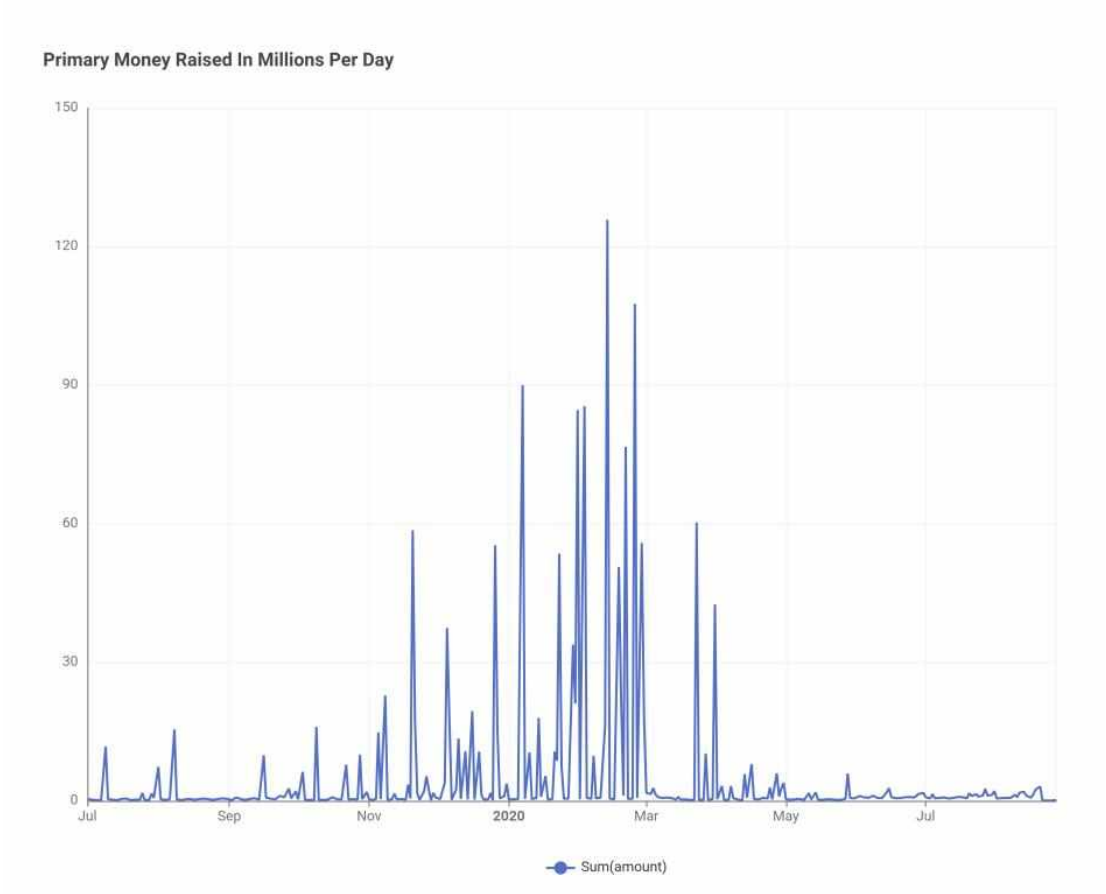
4. Reporting System

Analysis of the data was performed to gather insights on trends in spending before the election and which candidates received the most donations.

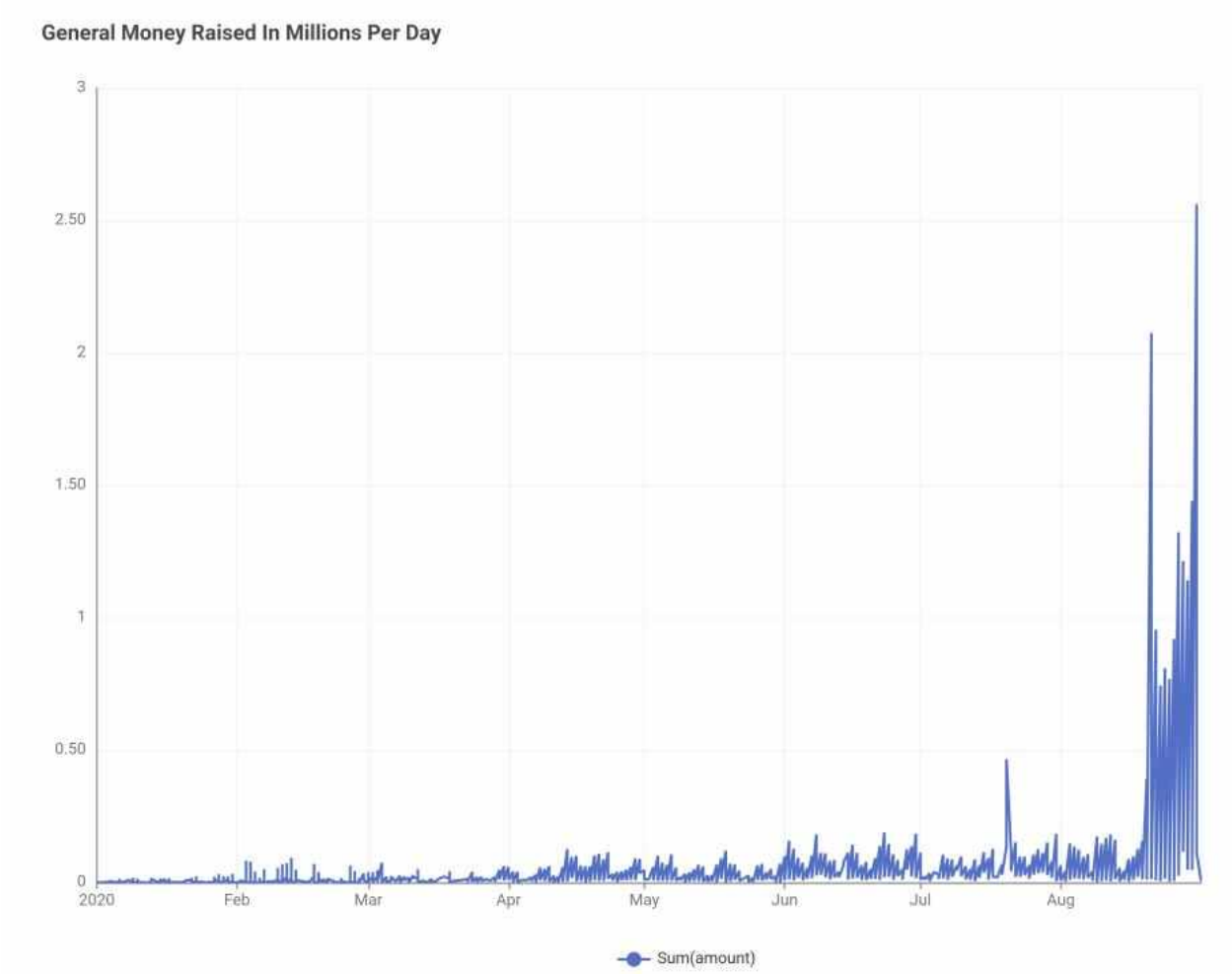
4.1 Relational Data Warehouse Implementation

Question 1: When were contributions made? Were contributions steady or did they mostly occur at a certain time?

For the initial analysis, the overall amount of money raised for each day was analyzed. Data showed an increasing total value of contributions in the days leading up to the primary election.

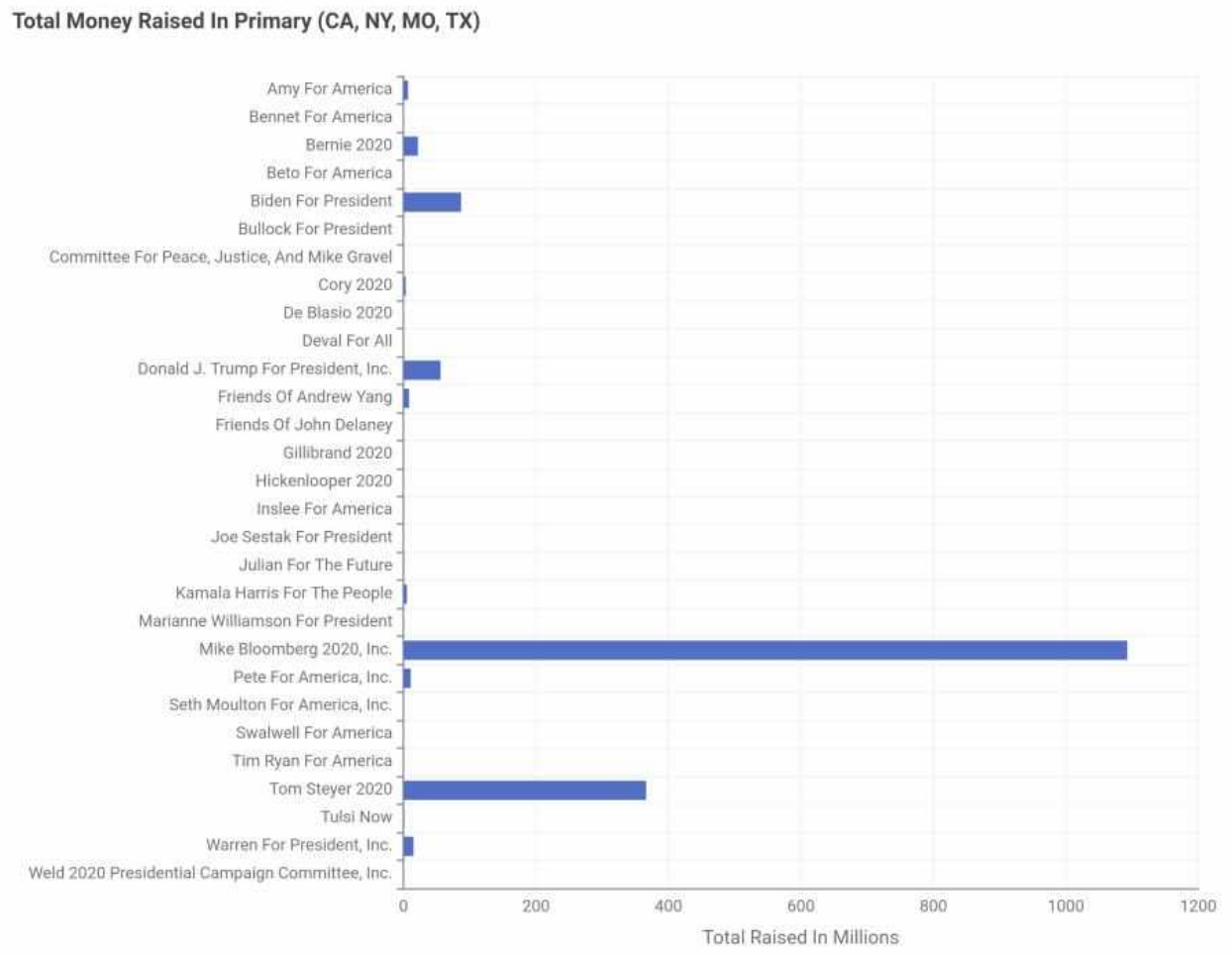


This same trend was also seen in donations leading up to the general election. There is a large difference in the amount of money per day donated though between the primary and general, with far larger amounts in the primary. This will be explained in the individual candidate graphs below.

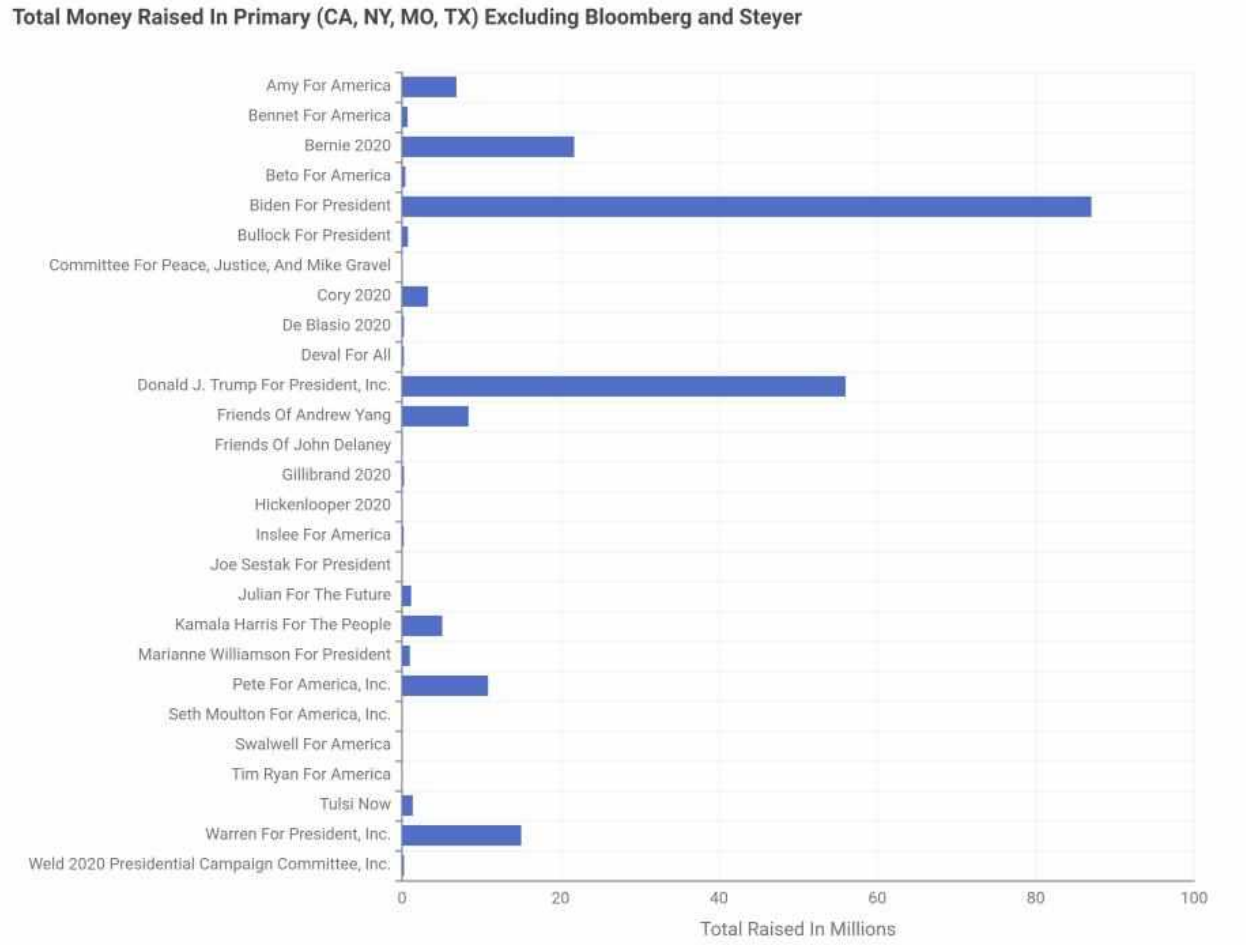


Question 2: Who raised the most money in the primary? Who raised the most in the general election? Did the person who raised the most win?

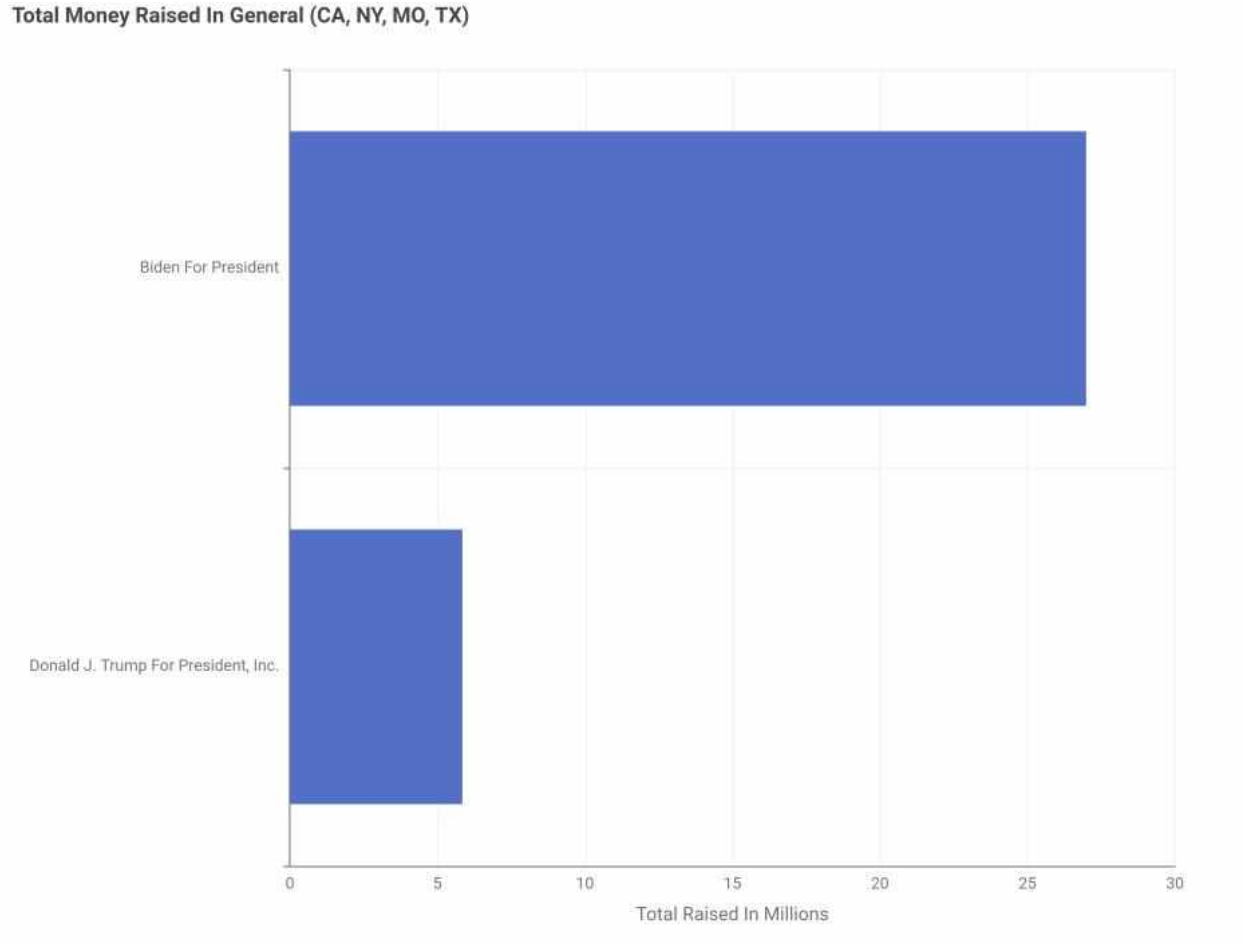
Mike Bloomberg raised the most money by far, well exceeding the other candidates although he did not secure the nomination. Steyer also raised a significantly higher amount than other candidates. The winner of the Democratic primary was Biden, who raised the third highest amount, which was far less than Bloomberg and Steyer.



A second graph was produced, excluding Bloomberg and Steyer, to better see the other primary candidates.

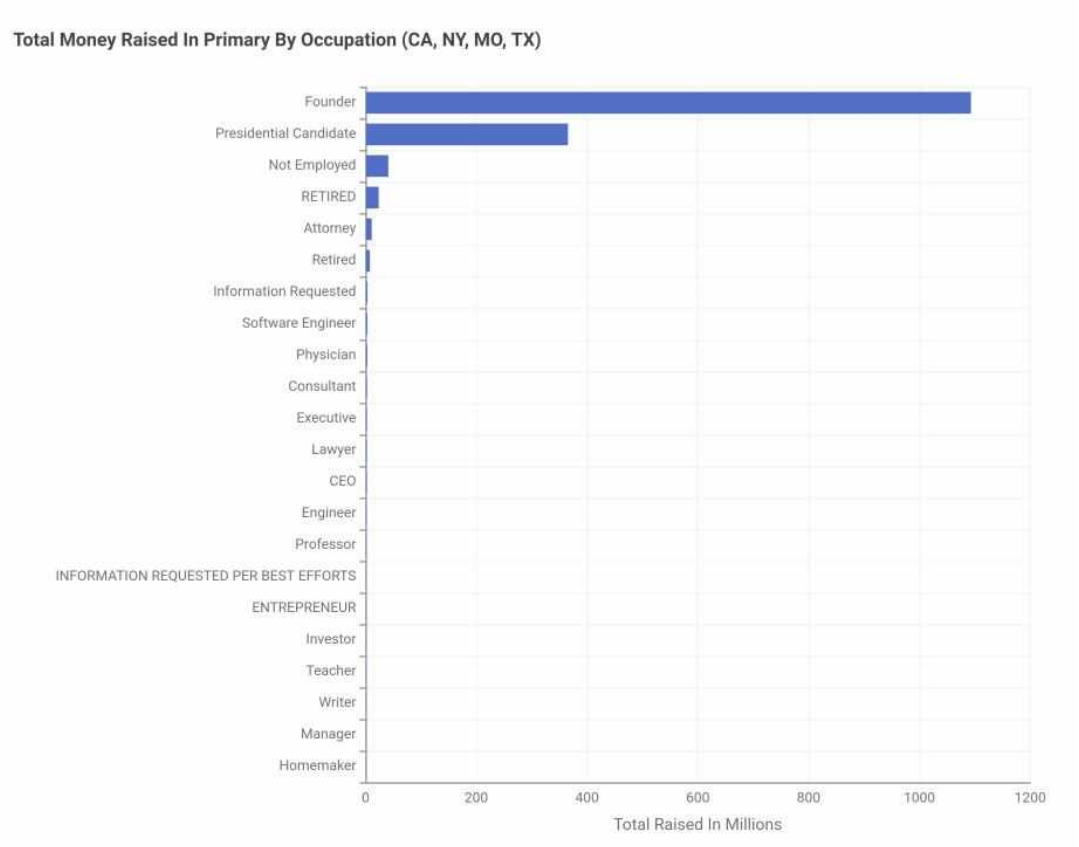


Joe Biden raised far more money, almost $20 million more in the four states, than Donald Trump in the general election. Joe Biden won the general election.

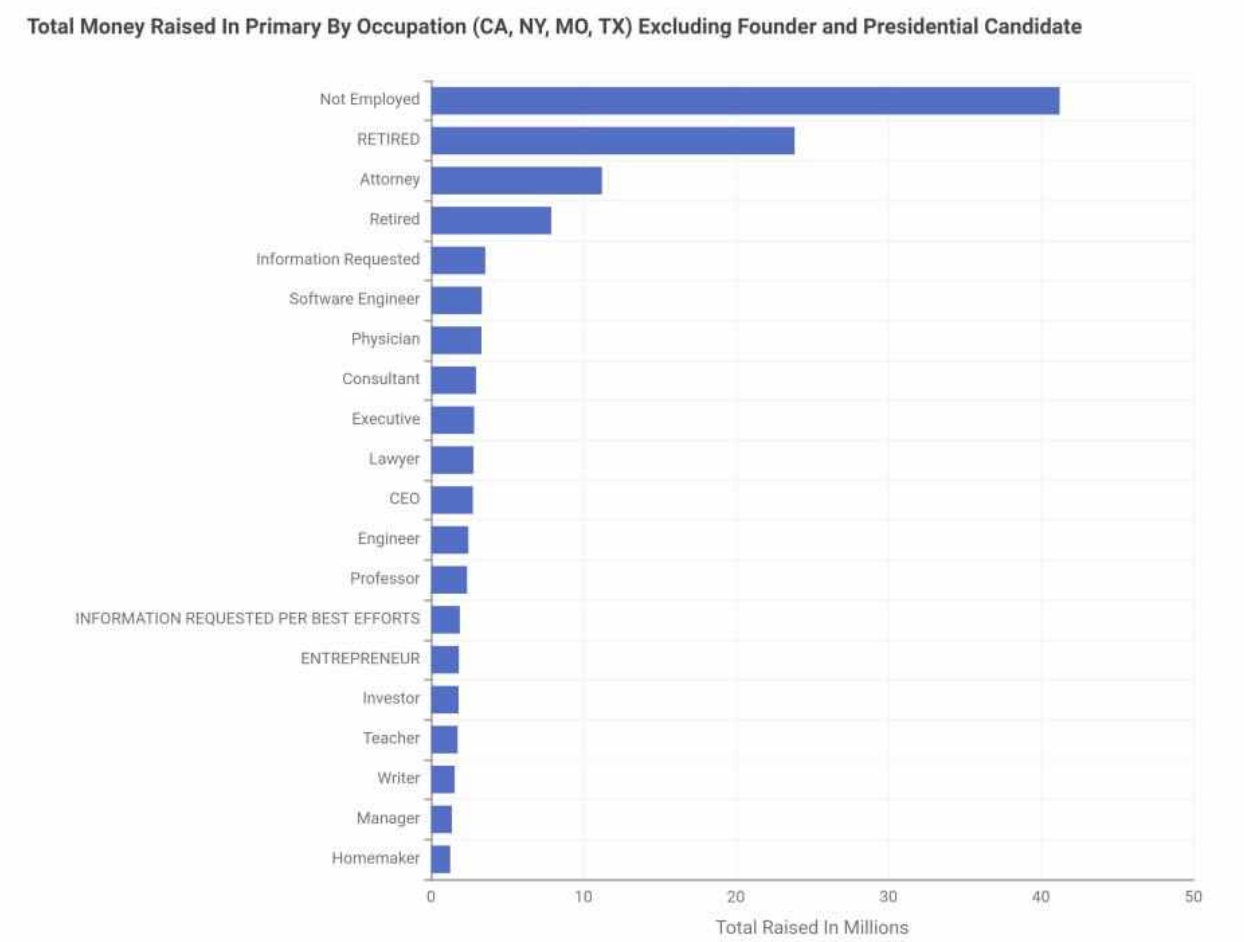


Question 3: What are some of the top occupations for donating money?

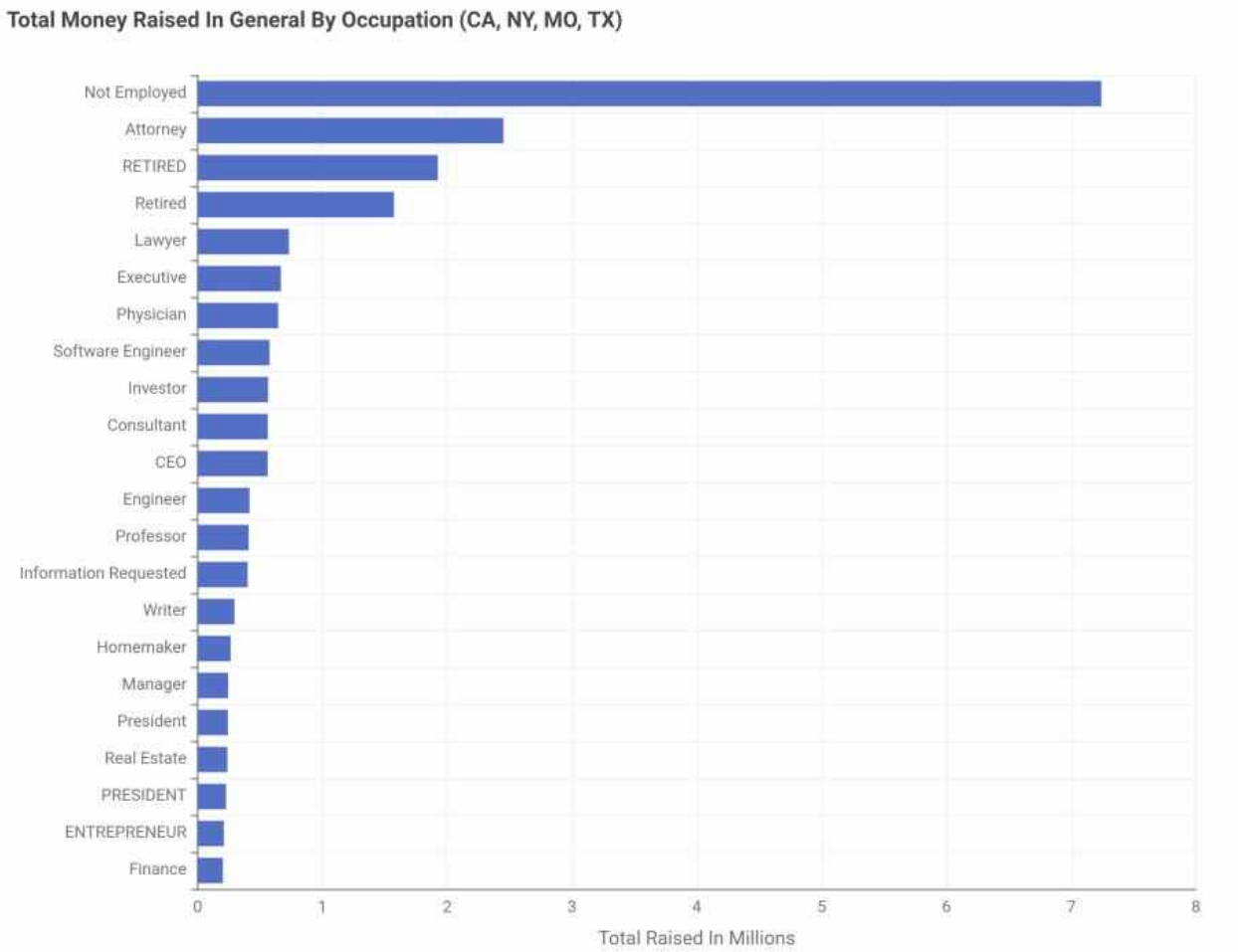
If we look at occupations, there are two that far exceed others in the amounts donated in the primary. Founder and Presidential Candidate. The two occupations are Bloomberg for Founder and Steyer for Presidential Candidate and should be removed.



Excluding those two results, we can see that not employed and retired donated the most. Note that there are two retired entries, one in all caps and the other lower case. It is also possible that not employed can also mean retired, indicating that the largest amount of donations come from retired workers. The highest contributions from working professions are from attorneys/lawyers, software Engineers, and physicians.

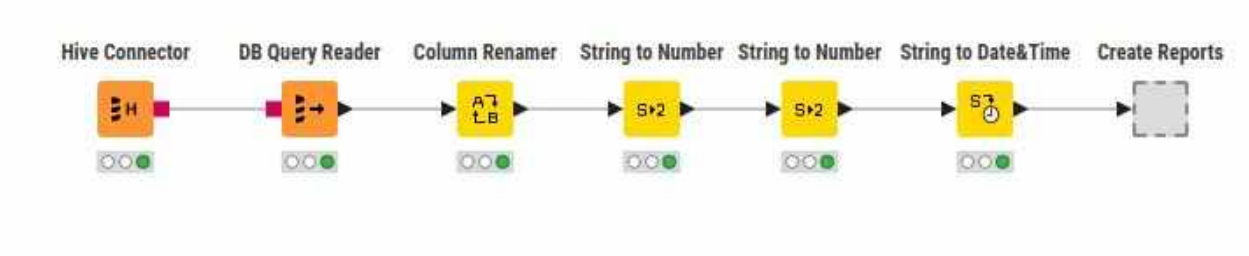


The general election showed similar trends, although this time executive was behind attorneys/lawyers, followed by physician and software engineer.



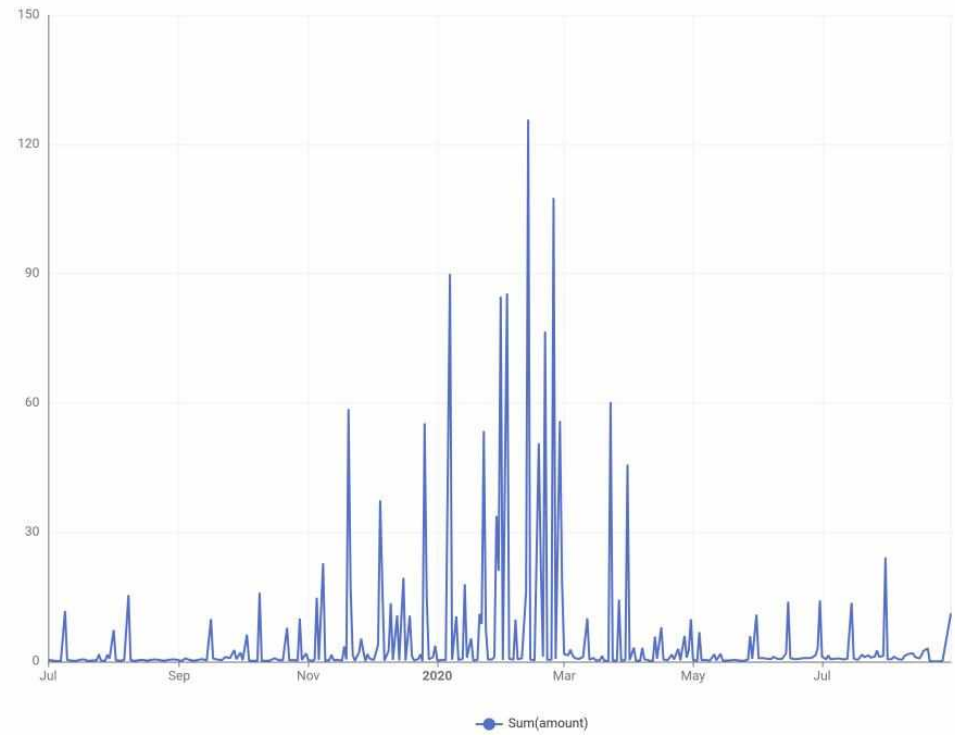
4.2 Hadoop Implementation

The data now stored in Hive can be analyzed using KNIME. Since the same data in the same format is stored in Hive as the data warehouse, minimal changes to KNIME need to be made to create the same charts as above. As is shown below, only a few data type conversions and column renames were needed to reuse the same reporting nodes.

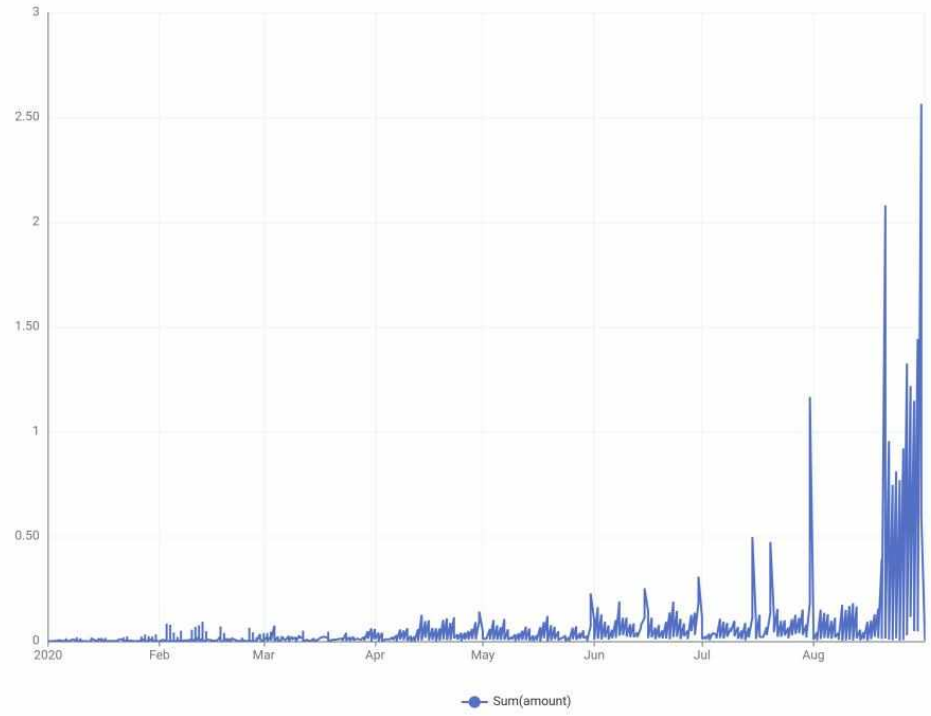


To adapt the data from Hive to the existing reporting nodes, four transformations were completed. Columns were renamed to match expected column names used in the data warehouse implementation. Zip codes were converted from strings to integers. Contribution amounts were converted from strings to doubles. Dates were converted from strings to the date data type. The data was then sent to the reporting nodes. As can be seen below, the Hadoop charts for question 1 appear similar to the data warehouse charts. There are a few additional peaks that are easily notable in the general election data, which could be a result of more data being deleted in the data warehouse solution compared to Hadoop. Checking the amount of rows in KNIME, 2,857,303 records were retrieved from the data warehouse, and 2,876,698 were retrieved from Hive. Since the charts produced are similar, only charts for question 1 are shown.

Primary Money Raised In Millions Per Day - Hadoop



General Money Raised In Millions Per Day - Hadoop



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

The Hadoop implementation was significantly easier to load the data. As the data had already been transformed using the Pig scripts, only a few transformations were needed to convert some of the columns into numeric or date format. The existing reporting nodes could be used as is once these transformations were completed.

Conclusions

Several questions were answered through analysis of the data. The analysis of the data showed that contributions increased significantly just before an election, and that Bloomberg raised the most money in the primary but lost to Biden. Biden raising the most in the general election and won the general. Both Bloomberg and Steyer contributed significantly to their own campaigns and had to be excluded to see results better from other candidates and occupations.

Regarding the implementation of the storage system and analysis, the data warehouse model involved significantly more work in create the structure of the schema and creating the nodes in KNIME necessary to store and retrieve the data. In contrast, the Hadoop implementation required only a small amount of code using Pig and Hive. While the data warehouse model may save on space due to eliminating some redundancy, the Hadoop solution was far quicker to implement than the warehouse model.