



California State University,  
Los Angeles



A

Project Report

On

**“Elasticsearch”**

Submitted By

Kumar, Rajnish [CIN: 304470392]

Under the Guidance of

Dr. Jiang Guo

For Quarter

Fall 2015

**DEPARTMENT OF COMPUTER SCIENCE**

## Table of Contents

Chapter No.		Topics	Page No.
1		Background	1
	1.1	What is Elasticsearch and its Background?	2
	1.2	Why Use Elasticsearch?	3
2		Current Status	4
	2.1	Current Status Introduction	5
3		Technical Issues	6
	3.1	Key Challenges	7
4		Solutions	6
	4.1	Data Load	8
	4.2	Standardization	11
	4.3	Pooling	24
	4.4	Scoring	25
	4.5	Calibration	26
	4.6	Clustering	27
	4.7	Inquiry	28
	4.8	Enrichment	30
	4.9	Bureau Run	32
	4.10	Bureau and ES Enrichment	33
	4.11	Technical Specification	36
	4.12	Elasticsearch setup	36
	4.13	Kopf	38
	4.14	Sample Codes	40
		Conclusion and Future Scope	

# **Chapter 1**

## **Background**

### 1.1 What is Elasticsearch and its Background?

**Elasticsearch** is a great tool for document indexing and powerful full text search used for **Big Data Analytics**. It's JSON based Domain Specific query Language is simple and powerful, making it the defector standard for search integration in any web app. The combination of storage and querying / aggregation services makes elasticsearch really special and distant from the "document storage only" tools.

Elasticsearch is in short can be thought as "**Search Engine Software**". It gives a disseminated, multitenant-competent full-content internet searcher with a relaxing web interface and pattern free JSON archives.

It has been created in Java and is discharged as open source under the terms of the Apache License. Elasticsearch is the second most prevalent endeavor web crawler.

It can be utilized to inquiry a wide range of reports, likewise gives versatile pursuit, and has close constant hunt and backings multitenancy.

Elasticsearch is dispersed, which implies that records can be separated into shards and every shard can have zero or more imitations. Every hub has one or more shards, and goes about as an organizer to delegate operations to the right shards. Rebalancing and directing are done naturally.

It utilizes Lucene and tries to make all elements of it accessible through the JSON and Java API. It backings faceting and permeating, which can be valuable for advising if new records match for enrolled inquiries.

Kibana is an open source program based investigation and quest dashboard for Elasticsearch. Kibana is a snap to setup and begin utilizing.

Kibana is composed altogether in HTML and JavaScript .It requires just a plain webserver, no extravagant server side parts.

Kibana is an incredible device for continuous information examination. . It endeavors to be anything but difficult to begin with, while likewise being adaptable and intense, much the same as Elasticsearch.

### 1.2 Why Use Elasticsearch?

These are three reasons why we use elasticsearch:

- ❑ It is very easy to get a toy instance of Elasticsearch running with a small sample dataset.
- ❑ Elasticsearch JSON based query language is much easier to master than more complex systems like Hadoop's Map Reduce.
- ❑ Application developers are more comfortable maintaining a second Elasticsearch integration over a completely new technology stack like Hadoop.

# **Chapter 2**

## **Current Status**

## 2.1 Current Status

A relational database engine is great at set based operations and has a very familiar and powerful querying syntax but we are increasingly facing data challenges where leveraging a relational model doesn't make as much sense. In building Raygun we have to deal with a constant stream of exception data (100's of millions per month) which we structure into errors and occurrences which have a simple 1:N relationship.

At first, using a relational model seemed to be a natural fit for this but over time we ran into three general problems:

1. Querying over the occurrence data was performing poorly despite being strictly index based. E.g. To find all parents where they have a child occurrence matching a particular criterion.
2. Producing aggregations of occurrence data was too costly to query in real time. Using a pre-aggregated bucket approach would add a massive overhead to our database size.
3. Searching the data using FTS would require indexing most of the raw payload which is costly both in terms of indexing time and database size and our preference was to use Lucene.

Surely we must just need a more powerful database cluster? More RAM, more CPU?

Rather than just blindly scaling, We always like to look around and see what everyone else is doing incase there is a better way. It turns out these are the type of problems that are handled nicely by ElasticSearch.

Given the product name you would have naturally concluded that ElasticSearch is a search server, so that sounds good if what you want is just full text search via an http endpoint, and this was how we initially deployed it with Raygun. Over time however we quickly realised that it offers very powerful and efficient querying which **makes it ideal for not just search but also quickly filtering sets and performing aggregations** (e.g. for analytics) over very large sets of data. Not just that it is incredibly simple to approach.

# **Chapter 3**

## **Technical Issues**



### 3.1 Key Challenges:

These are the key challenges:

- **Managing and Data Mining**

Managing the data is a very big problem today. In organization, we have to handle too much data and that time, mining is also needed but the challenges are nothing but to handle in effective way.

- **Filtering**

To fetch a list of parents, where they have a child occurrence matching a particular criterion, we want to first filter down to the occurrences matching the criteria and then group the results based on the parent error. This logically maps to a GROUP BY with a WHERE clause in the query in SQL.

This is nice and simple to achieve using Elasticsearch. Elasticsearch natively talks JSON and presents a very sane RESTful API for interaction. There are a number of native providers to deal with API interactions, which logically maps back to the underlying JSON would provide in a raw request.

This will find all occurrences where the property matches the value by using through a Filter and will then bucket the results into groupings based on the error they belong to by using the Term Facet. The end result is that we will have a list of all of the parents matching the predicate and a count of the number of occurrences under each error that match it.

- **Search**

When you hit a requirement like “search” for an application you face the unenviable task of having your application suddenly compete with Google or Bing in terms of speed and quality of results. Full text search providers in relational databases generally leave a lot to be desired. Using something more specialized such as Apache Lucene is the way to go which fits in perfectly for ElasticSearch since that’s what it uses, and that’s what it performs fantastically at.

# **Chapter 4**

## **Solutions**

## 4.1 Solutions

For all problems what we have in challenges, we can solve by using elasticsearch.

For that, I am going to elaborate the concepts what is basically used in elasticsearch and how it works.

### 4.1 Data Load

Data Load stage stacks the information into the end focus on that may be a basic delimited level record or an information distribution center. Contingent upon the necessities of the association, this procedure fluctuates generally. Some information distribution centers may include new information in a recorded structure at standard interims—for instance, hourly. To comprehend this, consider an information distribution center that is obliged to keep up deals records of the most recent year. This information stockroom overwrites any information more seasoned than a year with more current information. On the other hand, the section of information for any one year window is made in a verifiable way. The timing and degree to supplant or attach are vital outline decisions subject to the time accessible and the business needs.

Data Load stage interfaces with a database, the limitations characterized in the database blueprint and in addition in triggers enacted upon information load apply (for instance, uniqueness, referential uprightness, compulsory fields), which likewise add to the general information quality execution of the ETL procedure.

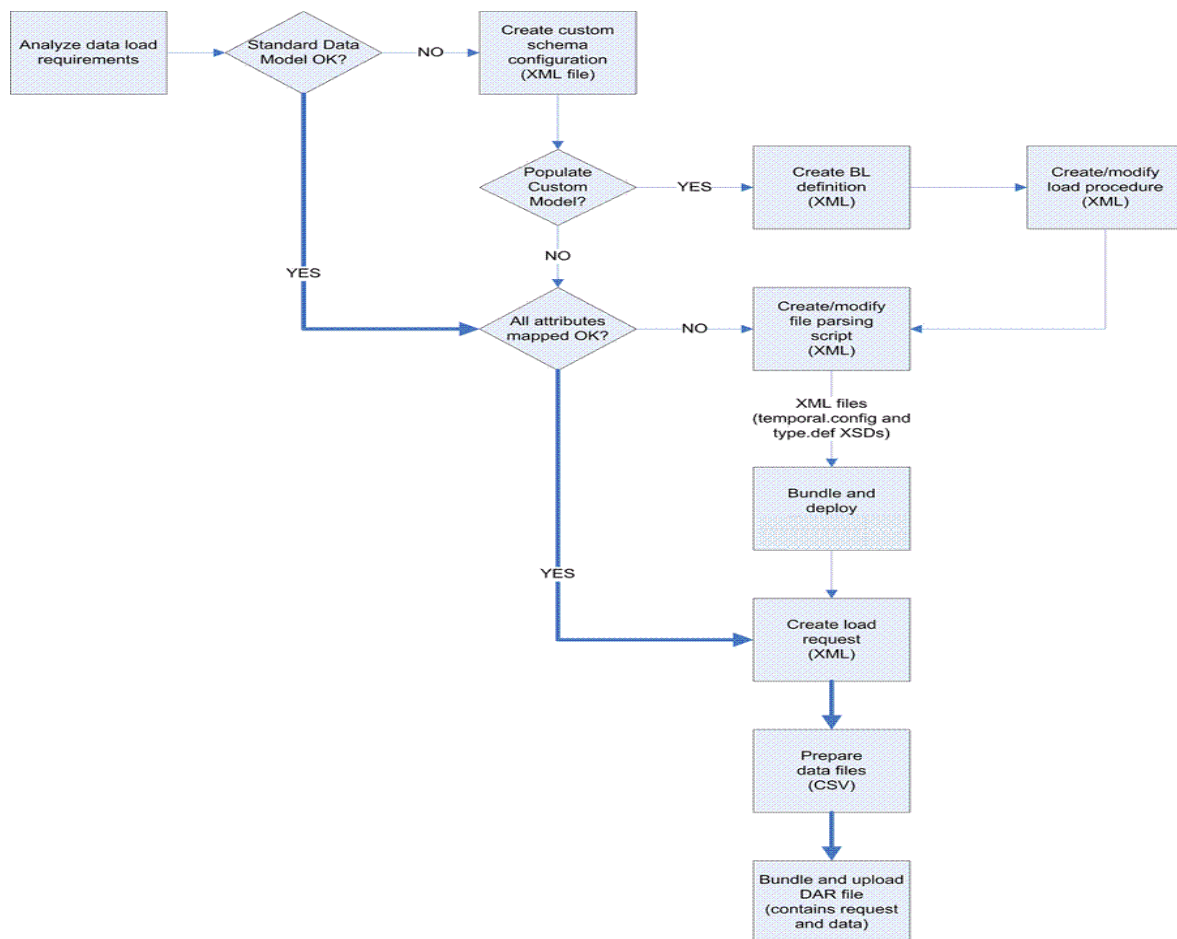
### Data Loading into My Elastic Search Source:

#### Step: 1 – Installation of Elasticsearch

1. Download Elasticsearch and unzip the latest Elasticsearch distribution.
2. Run the following command Line:  
`bin/elasticsearch -f -Xmx2g -Xms2g -Des.http.max_content_length=1.5gb`
3. Run:  
`curl -X GET http://localhost:8000/`

## Step: 2 – Loading Data into Elastic Search

1. To create your index  
⇒ `curl -s -XPUT http://localhost:8000/INDEX_NAME/`
2. To load the schema  
⇒ `curl -s -XPOST http://localhost:8000/INDEX_NAME/TYPE_NAME/_mapping--data-binary@SCHEMA_FILE`
3. To load the data  
⇒ `curl -XPOST 'http://localhost:8000/INDEX_NAME/TYPE_NAME/_bulk' --data-binary @DATA_FILE`



(Fig. 4.1: Loading Data)

## 4.2 Standardization

The data that is provided to a Credit-Bureau company, has passed many hands and is collected from all the corners of the world. This (huge) data is not in the proper/standard format. While working on/with data, it is very important that the data is of set-format and standard. The cut-throat competition in the Banking and Insurance sector has led to a mad race to rope in customers first and verify their credentials later, which inadvertently, leads to non-standard and unformatted data. The incompetence of the agents involved at the root level, also adds to this chaos.

As a result, the data provided to us, to work on/with, is unworkable, as is, and needs some Pre-Processing. This is where Data Standardization comes in to the rescue. The unkempt, unruly, non-standard data is worked on/with and processed, to make it usable.

Data Standardization involves several custom pre-processing steps, depending on the locale from which the data is collected. The pre-defined data formats of an individuals' identity are taken and used as a guideline to process the data given. After processing, the data is passed on to the next level in the Credit-Bureau's chain of commands.

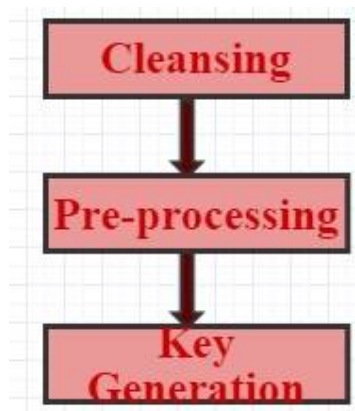
Standardization is the process of developing and implementing technical standards. Standardization can help to maximize compatibility, interoperability, safety, repeatability or quality. It can also facilitate commoditization of formerly custom processes.

Standardization of data is a means of changing reference data set to a new standard. It is a key part of ensuring data quality.

The motivation behind Data Standardization is to make your information steady and clear. Predictable is guaranteeing that the yield is dependable so that related information can be recognized utilizing regular wording and configuration. Clear is to guarantee that the information can be effortlessly comprehended by the individuals who are not included with the information upkeep process.

Standardization process is carried out in three ways:

1. Cleansing
2. Preprocessing
3. Key Generation



(Fig. 4.2: Standardization Process)

#### 4.2.1 Cleansing

Cleansing is one of the step of Standardization process which removes multiple white-spaces and special characters.

It is a process of detecting and correcting corrupt or inaccurate records from a record set, table or database.

The inconsistency detected or removed may have been originally caused by user entry errors or by different data dictionary definitions of similar entities.

#### 4.2.2 Preprocessing

Preprocessing omits dictionary related rules such as prefix and suffix. It is second step of standardization process in which the salutations of name, Country code of phone number and special tokens are removed. The prefixes are configurable and can be changed according to the need of Bureau.

Real world data is often incomplete or inconsistent and lacking in certain behaviors or trends, and is likely to contain many errors.

Data Preprocessing is a proven method to resolving such issues. It prepares raw data for further processing.

### 4.2..3 Key Generation

Generally, three algorithms are used for generating key. These are as follows:

1. Metaphone 3
2. Colognephonetic
3. Refined Soundex

These Algorithms are used for generating unique key

#### 1. Metaphone 3 Algorithm:

Metaphone is a phonetic algorithm, utilized for indexing words by their English elocution. It generally enhances the Soundex calculation by utilizing data about varieties and irregularities in English spelling and elocution to deliver a more exact encoding, which improves employment of coordinating words and names which sound comparative.

The Double Metaphone phonetic encoding calculation is the second era of this calculation.

It is called "Twofold" on the grounds that it can return both an essential and an auxiliary code for a string; this records for some equivocal cases and additionally for numerous variations of surnames with basic family line. For instance, encoding the name "Smith" yields an essential code of SM0 and an auxiliary code of XMT, while the name "Schmidt" yields an essential code of XMT and an optional code of SMT—both have XMT in like manner.

**Metaphone 3** further enhances phonetic encoding of words in the English dialect, non-English words well known to first names and family names. It enhances encoding for legitimate names specifically to an impressive degree. Designers can likewise now set changes in code to bring about the calculation to encode Metaphone keys 1) **considering non-starting vowels**, and additionally 2) **encoding voiced and unvoiced consonants in an unexpected way**. This permits the outcome set to be all the more firmly centered if the engineer finds that the list items incorporate an excess of words that don't take after the pursuit term firmly enough.

## 2. Calogne phonetic Algorithm:

```
public class ColognePhonetic
    extends Object
    implements StringEncoder
```

⇒ Encodes a string into a Cologne Phonetic value.

## 3. Refined Soundex Algorithm:

Refined Soundex is a phonetic calculation for indexing names by sound, as maintained in English. The objective is for homophones to be encoded to the same representation with the goal that they can be coordinated in spite of minor contrasts in spelling. The calculation predominantly encodes consonants; a vowel won't be encoded unless it is the first letter. Soundex is the most broadly known of every single phonetic calculation (to some extent in light of the fact that it is a standard component of famous database programming, for example, DB2, PostgreSQL, MySQL, Ingres, MS SQL Server and Oracle) and is frequently utilized (mistakenly) as an equivalent word for "phonetic calculation" Improvements to Soundex are the premise for some cutting edge phonetic calculations.

As such, we, at **CRIF High Mark**, strive to provide standard data. Our standardization process involves

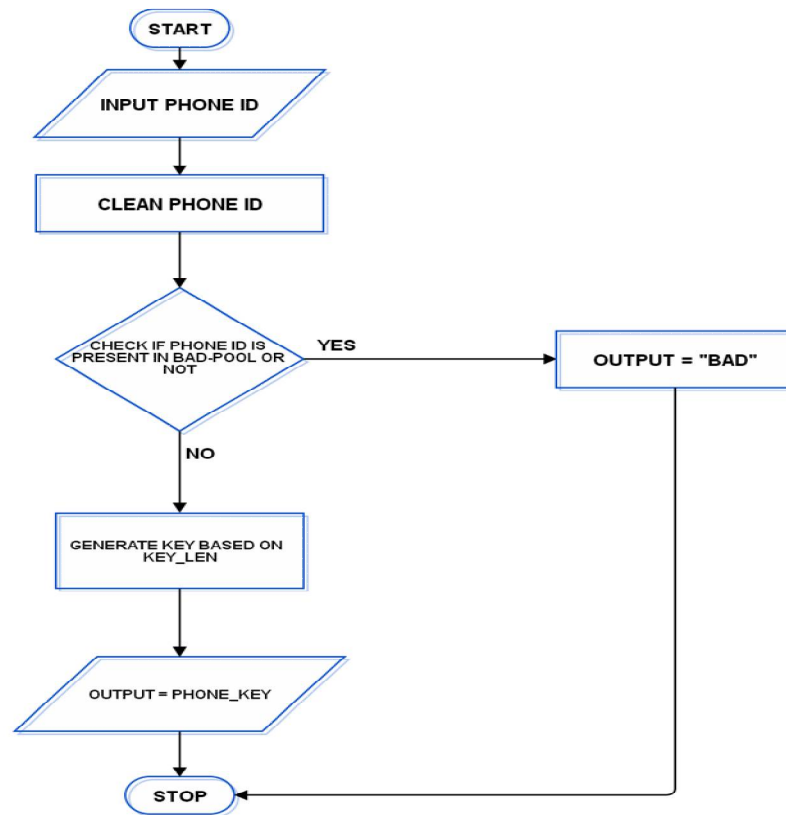
- Standardization of Consumers' Phone Number
- Standardization of Consumers' ID
- Standardization of Consumers' Date of Birth
- Standardization of Consumers' Name
- Standardization of Consumers' Address

Now, let us discuss each one of the above processes in detail.



**Standardization of Consumers' Phone Number: -**

Consider the flowchart below:-



(Fig. 4.3: Standardization of Consumer's Phone Number)

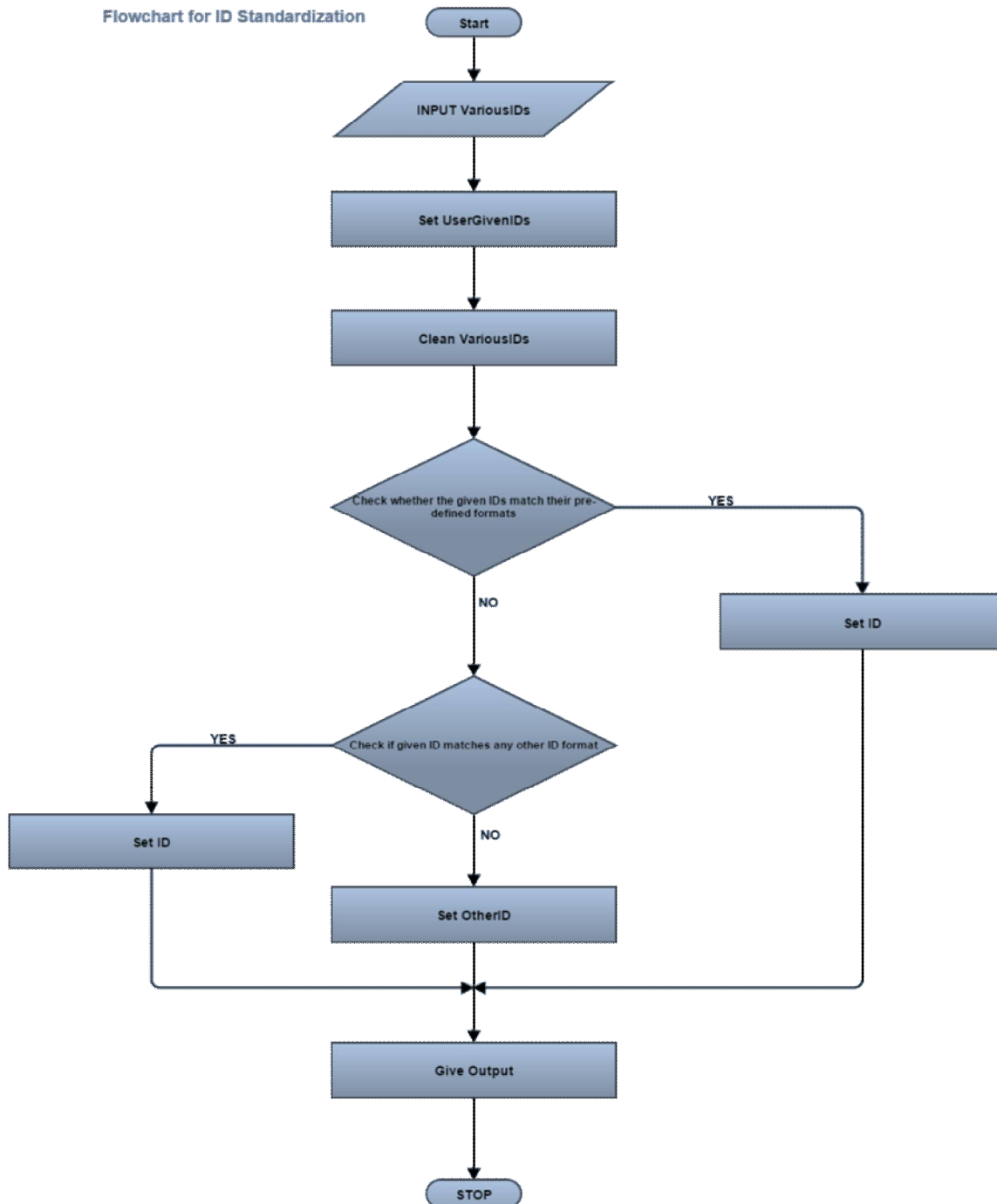
This simple flowchart explains how we process non-standard Consumer Phone Number. First, the given phone number is stripped of all the special characters, if any, that may be present in it.

Then, we check and remove any prefixes (country codes and others mentioned in the code), if any. The prefixes are configurable and can be modified according to the need of the Bureau. After the prefixes are removed, this number is checked against a pool of known pool of bad and/or invalid/restricted phone numbers.

If the given phone number is present in the Bad Pool, the phone number is removed from further processing, else it is passed on to the next process. The next process will generate a key of length that is configurable and can be changed according to the need of the Bureau.

**Standardization of Consumers' Different IDs:-**

Consider the flowchart below:-



(Fig. 4.4: Standardization of Consumer's ID)

The flowchart above gives an idea of what's happening in the Standardization of Consumer's Different ID part of the Data Standardization process.

Firstly, various Consumer's ID are taken and stored as input. These input IDs are cleaned of any special characters except for those that are expressly allowed by the programmer.

We have a pool of known bad IDs, which we have created after taking many things in account. After cleaning, we check whether the clean input IDs are present in the pool of bad IDs or not. If the clean input ID is present in the Bad-Pool, we discard it and process other IDs, else, we carry on with the process.

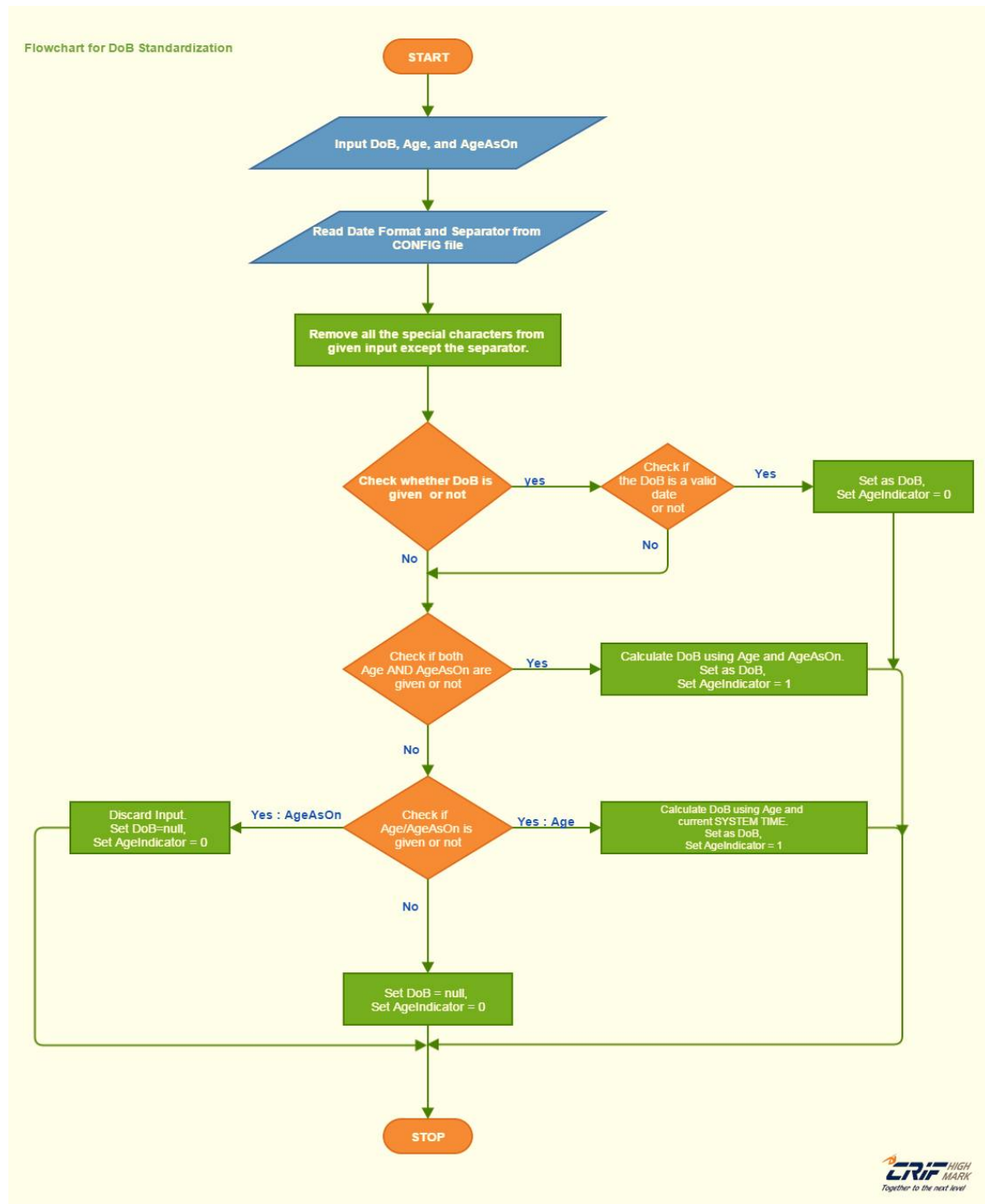
We have created a Format file which consists of all the valid ID formats defined as RegEx.

When an ID is to be checked, it is first of all, checked against the RegEx of the ID where the input ID is. If the input ID matches the RegEx of the ID present in Format file, then that ID is set in the Process appropriately, else the input ID is matched against all the other ID format RegEx. If a match is found, the input ID is set as the ID for which it matches the RegEx. If no match is found, we move the input ID in the Other ID set.

This part of the Data Standardization process returns all the IDs in Standard form to higher processes in the Bureaus Chain of Commands.

### Standardization of Consumers' Date of Birth:-

Consider the flowchart below:-



(Fig. 4.5: Standardization of Consumer's Date of Birth)

The flowchart above hints at what's happening in the Standardization of Consumer's Date of Birth part of the Data Standardization process.

Firstly, the given DOB, Age and AgeAsOn is taken as input. Then, a config file, which contains the Date Format of the locale and Separator, is read. Then, all the special characters, except the read Separator, are removed from the given input.

We, now, check whether the given input contains a DOB or not. If input contains a DOB, then we check whether the DOB is a valid Date or not. If the DOB is not valid, the input is treated as not having a DOB. If the DOB is valid, then the user given DOB is set as standard DOB, the Age-Indicator is set as equal to 0 and we move to another input. The Age Indicator is used further up in the Bureau's process to determine whether the data in the database is user given or computer generated, which in turn determines its reliability when a query is hit.

If the input does not contain a DOB, we check whether the input has both Age and AgeAsOn or not. If both Age and AgeAsOn are present in the input, then standard DOB is calculated using them and Age-Indicator is set as equal to 1. If, both Age and AgeAsOn, are not present, then we check if any one of them is present in the user given input.

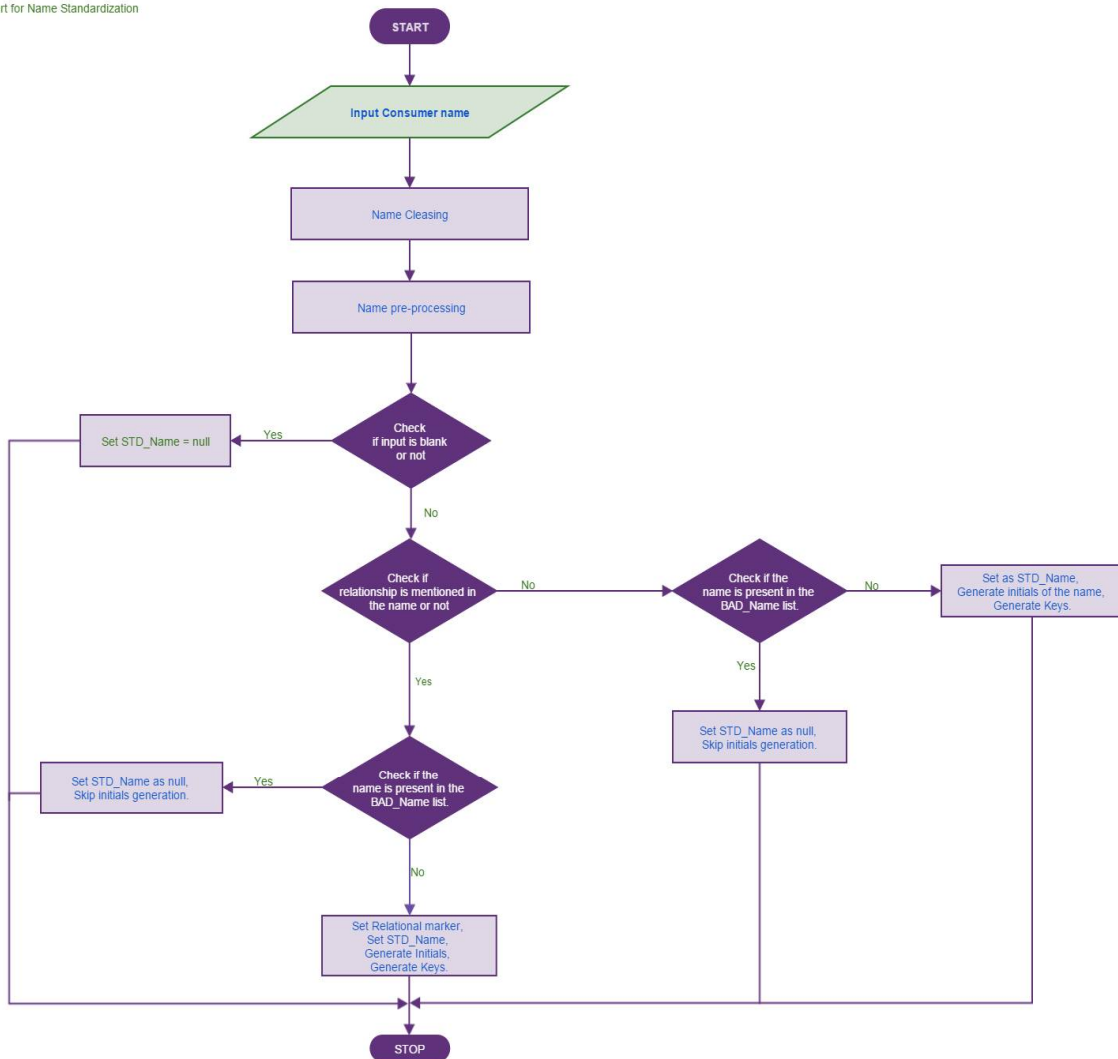
If the input contains only AgeAsOn, we discard the input, set standard DOB as equal to null, set Age-Indicator as equal to 0 and move to another input. If the input contains only Age, then we calculate the standard DOB using given Age and current system time and set Age Indicator as equal to 1.

If, neither Age nor AgeAsOn is present in the input, we set standard DOB as null and set Age Indicator as equal to 0 and move to another input.

**Standardization of Consumers' Name:-**

Consider the flowchart below:-

Flowchart for Name Standardization



(Fig. 4.6: Standardization of Consumer's Name)

The flowchart above briefly explains the working of the process of Name Standardization.

Firstly, the input name is cleansed of any special character or numeral present in it. Then, in the pre-processing stage, stage, salutations and special tokens, if any, are removed.

Now, the input is checked whether to blank or not. If the input, after cleansing and pre-processing, is found to be blank then STD\_Name is set equal to null. If the input is not blank then it is sent for further processing.

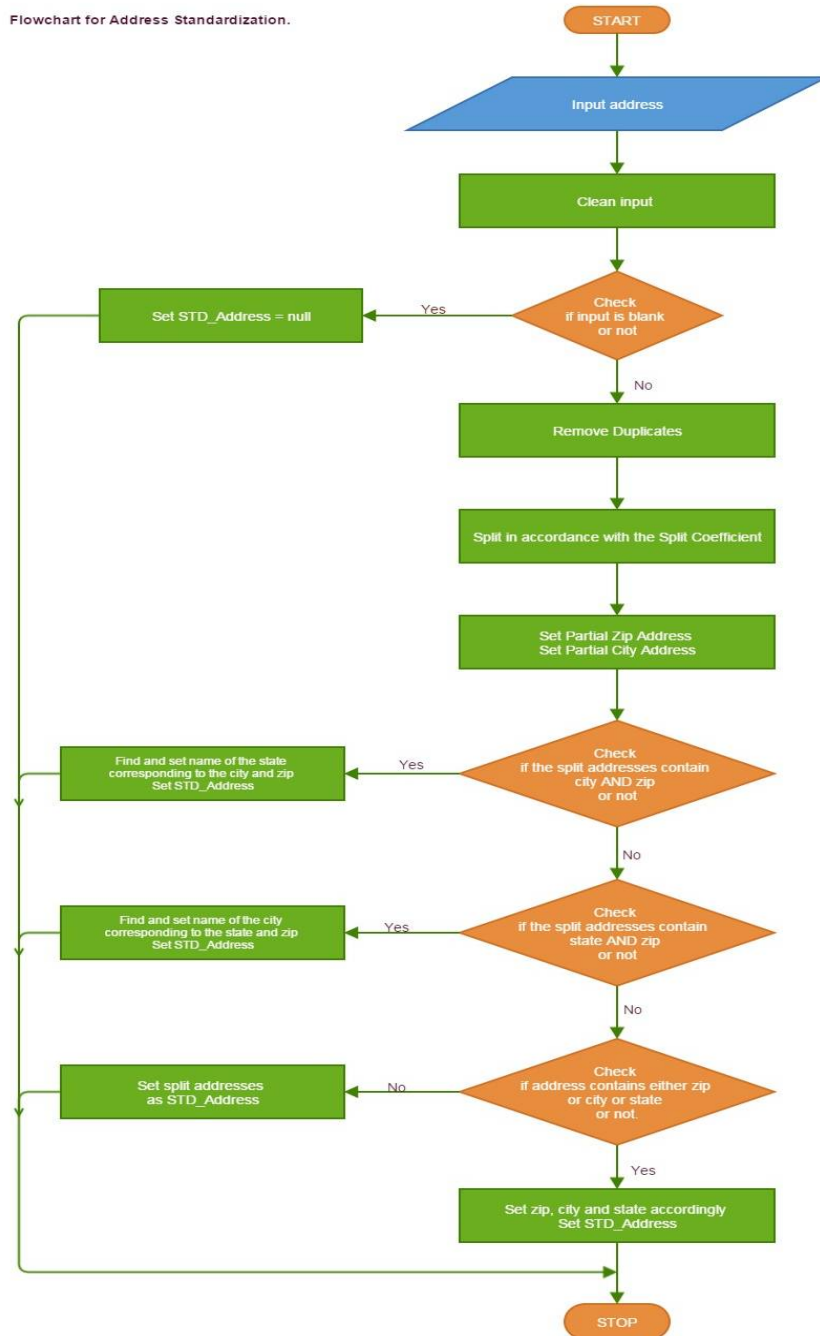
After the Blank-Check, the input is checked for any relationship markers, s/o, c/o, d/o, etc. If no relationship marker is present, then we check whether the name is present in the Bad\_Name list or not. If the name is found in the Bad\_Name list STD\_Name is set as null and Name-Initials generation is skipped. If the name is not found in the Bad\_Name list, the name is set as STD\_Name and Name-Initials are generated and different phonetic keys are generated.

If there is a relationship marker present in the input, then we set the Relationship marker and check again for bad names against the pool of bad names. If name is matched against a bad name in the Bad\_Name pool, then we set STD\_Name as equal to null and skip Name-Initials generation. If, the name is not matched against any name present in the Bad\_Name list, then it is set as STD\_Name, Name-Initials are generated and different phonetic keys for the name are generated.

The process, thus, ends and the input name is standardized and another raw input name is taken as input for standardization.

### Standardization of Consumers' Address:-

Consider the flowchart below:-



(Fig. 4.7: Standardization of Consumer's Address)



The flowchart above shows a simplified and brief version of what is happening inside the Address Standardization process.

Firstly, the data is input. This data is cleaned of any special characters and multiple white-spaces. Then it is checked to be blank or not. If the data after cleansing is blank STD\_Address is set as equal to null else the data goes on for further processing.

The data is then checked for any duplicate entries. If any duplicates are found, they are removed and the data is sent further for processing.

Then, the data is split according to a parameter called “*splitAddressPercentage*”. This step generates two split addresses. Normally, one of the split addresses would contain city name and the other would contain the zip code. These partial addresses are sent further for processing.

Now, the data is checked to contain both city AND zip code. If both are present, then corresponding state is found and STD\_Address is set, else the data is pushed further.

Then the data is checked to contain both state AND zip. If both are found then the corresponding city is found out and STD\_Address is set, else the data is pushed down the flow.

If none are found then, the split addresses are set as STD\_Address, otherwise the city or state or zipcode are sent accordingly and the STD\_Address is thus set.

### 4.3 Pooling

We have used Pooling process to utilize a shared access to a database.

Pooling is a method to permit various clients to make utilization of a stored arrangement of shared and reusable association articles giving access to a database.

Opening/Closing database associations is a costly process and consequently association pools enhance the execution of execution of summons on a database for which we keep up association objects in the pool. It encourages reuse of the same association article to serve various customer solicitations. Each time a customer solicitation is gotten, the pool is looked for an accessible association object and it's exceedingly likely that it gets a free association object. Something else, either the approaching solicitations are lined or another association item is made and added to the pool (contingent on what number of associations are as of now there in the pool and what number of the specific usage and setup can bolster). When a solicitation wraps up an association protest, the item is offered back to the pool from where it's doled out to one of the lined solicitations (taking into account what booking calculation the specific association pool usage takes after for serving lined solicitations). Since the majority of the solicitations are served utilizing existing association questions just so the association pooling methodology cuts down the normal time needed for the clients to sit tight for setting up the association with the database.

It's regularly utilized as a part of an online venture application where the application server handles the obligations of making association articles, adding them to the pool, doling out them to the approaching solicitations, taking the utilized association items back, returning them back to the pool, and so on. At the point when a dynamic website page of the online application explicitly makes an association (utilizing JDBC 2.0 pooling supervisor interfaces and calling `getConnection()` strategy on a `PooledConnection` object to the database and closes it after utilization then the application server inside gives an association object from the pool itself on the execution of the announcement which tries to make an association and on execution of the announcement which tries to close the association, the application server basically gives back the association back to pool.

## 4.4 Scoring

Scoring is one of the step of standardization process in which score is generated corresponding to the given rule code .It decides the score based on Gap Starting and Gap extension.

Scoring consequently institutionalizes or focuses the DATA= variables, taking into account data from the first variables and investigation from the SCORE= information set.

On the off chance that the SCORE= scoring coefficients information set contains perceptions with \_TYPE\_='MEAN' and \_TYPE\_='STD', then PROC SCORE institutionalizes the crude information before scoring. For instance, this kind of SCORE= information set can originate from PROC PRINCOMP without the COV choice.

In the event that the SCORE= scoring coefficients information set contains perceptions with \_TYPE\_='MEAN' yet \_TYPE\_='STD' is truant, then PROC SCORE focuses the crude information (the methods are subtracted) before scoring. For instance, this kind of SCORE= information set can originate from PROC PRINCOMP with the COV choice.

In the event that the SCORE= scoring coefficients information set does not contain perceptions with \_TYPE\_='MEAN' and \_TYPE\_='STD', or on the off chance that you utilize the NOSTD choice, then PROC SCORE does not focus or institutionalize the crude information.

In the event that the SCORE= scoring coefficients are gotten from perceptions with \_TYPE\_='USCORE', then PROC SCORE "institutionalizes" the crude information by utilizing the uncorrected standard deviations distinguished by \_TYPE\_='USTD', and the methods are not subtracted from the crude information. For instance, this kind of SCORE= information set can originate from PROC PRINCOMP with the NOINT choice. For more data about \_TYPE\_='USCORE' scoring coefficients in TYPE=UCORR or TYPE=UCOV yield information sets, see Appendix A, Special SAS Data Sets.

We can utilize PROC SCORE to score the information that were additionally used to create the scoring coefficients, albeit all the more regularly, scoring results are straightforwardly acquired from the OUT= information set in a system that registers scoring coefficients. At the point when scoring new information, it is critical to understand that PROC SCORE expect that the new information have pretty nearly the same scales as the first information. For

instance, on the off chance that you indicate the COV choice with PROC PRINCOMP for the first examination, the scoring coefficients in the PROC PRINCOMP OUTSTAT= information set are not proper for institutionalized information. With the COV choice, PROC PRINCOMP won't yield \_TYPE\_='STD' perceptions to the OUTSTAT= information set, and PROC SCORE will just subtract the method for the first (not new) variables from the new variables before increasing. Without the COV choice in PROC PRINCOMP, both the first variable means and standard deviations will be in the OUTSTAT= information set, and PROC SCORE will subtract the first variable means from the new variables and separation them by the first variable standard deviations before duplicating.

When all is said in done, techniques that yield scoring coefficients in their OUTSTAT= information sets give the important data to PROC SCORE to focus the fitting institutionalization. On the other hand, on the off chance that you utilize PROC SCORE with a scoring coefficients information set that you built without \_TYPE\_='MEAN' and \_TYPE\_='STD' perceptions, we may need to do the pertinent focusing or institutionalization of the new information first. On the off chance that you do this, you must utilize the methods and standard deviations of the first variables—that is, the variables that were utilized to produce the coefficients—not the methods and standard deviations of the variables to be scored.

## 4.5 Calibration

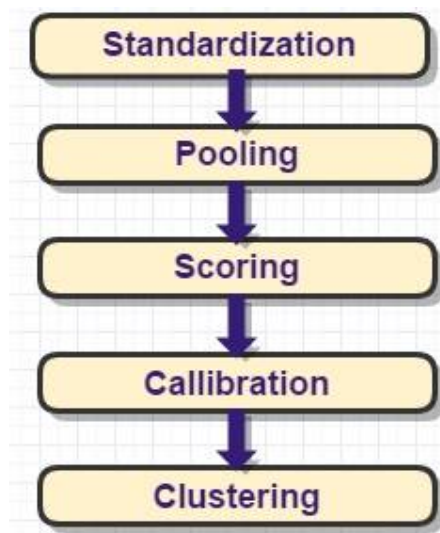
- In this Step, Match Levels are defined .There are certain Match key Levels, according to which we are able to decide that which files comes under which match key levels.
- L0- Not present in inquiry.
- L1- Exact Match
- L2- 1+ 2 Digit mismatch. It is also called as high match.
- L3- 1+ 2+ 3 Digit mismatch. It is also called as low match
- L4 – Perfect mismatch.It is also called no match.

## 4.6 Clustering

Clustering is the undertaking of collection an arrangement of items in a manner that questions in the same gathering (called a group) are more comparable (in some sense or another) to one another than to those in different gatherings (bunches). It is a fundamental undertaking of exploratory information mining, and a typical system for factual information investigation, utilized as a part of numerous fields, including machine learning, example acknowledgment, picture examination, data recovery, and bioinformatics.

Clustering is used for loading, cleaning and finally getting the data.

So, Overall Procedure of Data Loading can be seen by the below figure:



(Fig. 4.8: Data Loading Process)

## 4.7 Clustering

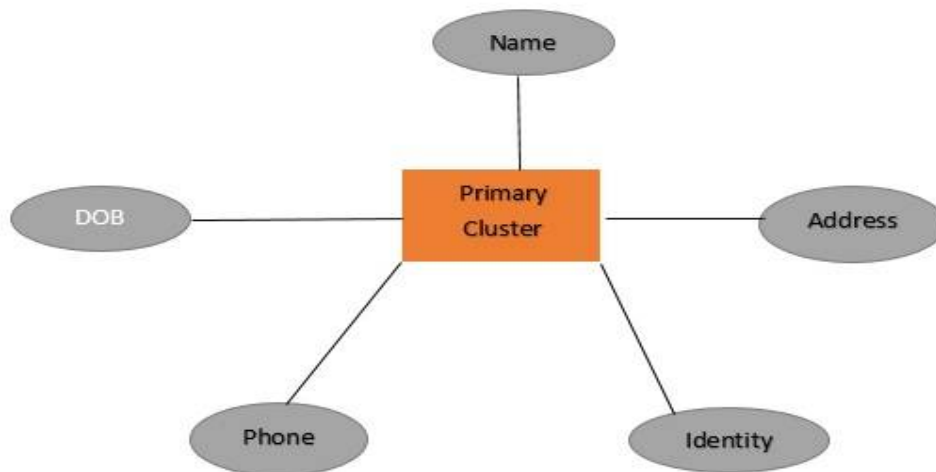
Inquiry is the process that has the aim of augmenting knowledge, resolving doubt or solving a problem. When the file is loaded, it is in the form of clusters. Clusters are the responsible to getting data and after cleaning, it is loaded into main system after client request.

Inquiry has been subcategorized into two parts:

### Primary Cluster:

It is the cluster table which holds only unique details of a consumer.

All consumer having unique name, address, DOB, phone and identity are stored in a primary cluster.



(Fig. 4.9: Primary Cluster Data)

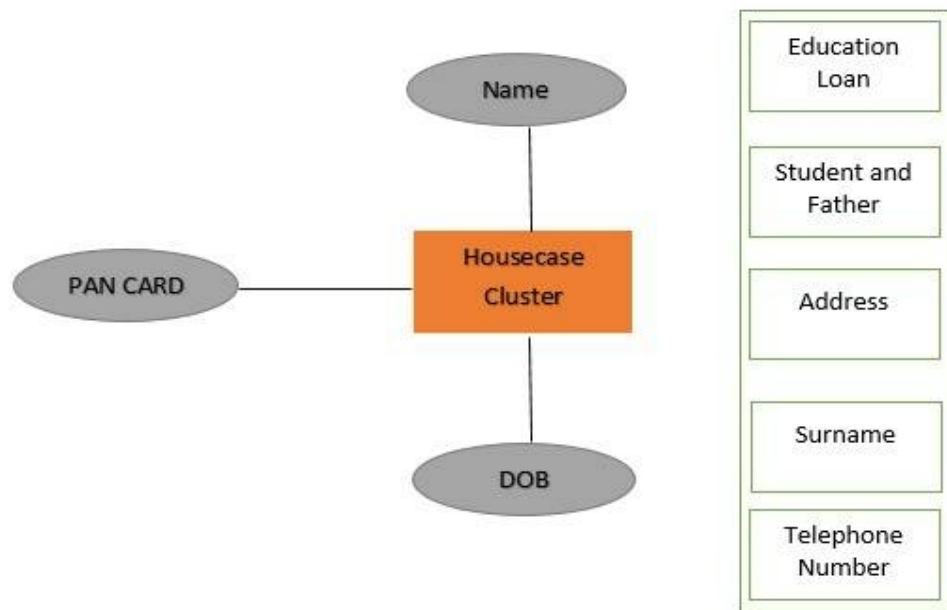
Examples of Primary Clusters are:

- Verification of identity by a financial institution such as a credit union or stock brokerage
- Renting a Car
- Getting an Internet Connection
- Opening a saving account
- Checking your own credit score

**Housecase Cluster:**

It is the cluster which having the uncommon details of the consumer.

For example, In case of bank loan. If I want to take Education Loan from a bank and I am a dependent on my father / mother. Some details such as Surname, Telephone Number and address are same from dependent except some values such as first name (i.e. mine and dependent), mobile number, pan card details, date of birth, then first name, pan card number, date of birth will be kept in household clusters.



(Fig. 4.10: Housecare Clusters)

Examples of Housecare Clusters are:

- Getting a Bank Loan with the help of Sponser (may be father, mother, brother) such as Student Education Loan, Auto Loan, Business Loan etc.
- Applying for a Credit Card
- Getting a cell phone on contract
- Requesting a credit limit increase
- Applying for a mortgage

## 4.8 Enrichment

**Enrichment** can be a helpful instrument in the push to increase the value of and enhance the nature of your association's information. It depicts the procedure of improving information as the execution of an arrangement of routines and structures for refining and upgrading data, which may appear like a dubious idea.

The objective of advancing information is to make it a more significant resource for receive more in return, to accomplish more with it, to get to it all the more effectively and to be more proactive in its utilization all without discernibly expanding expenses or dangers.

It is a general term that alludes to procedures used to upgrade, refine or generally enhance crude information. It makes information a profitable resource for advanced business or undertaking.

The Data Integration procedure is customarily considered in three stages: concentrate, change, and burden. Setting aside the regularly examined request of their execution, "concentrate" is hauling information out of a source framework, "change" means approving the source information and changing over it to the wanted standard and burden means putting away the information at the destination.

An additional step, Data "**Enrichment**", has recently emerged, offering significant improvement in business value of integrated data. Applying it effectively requires a foundation of sound data management.

The most obvious enrichment example is address correction. When we enter our address on some sites, the site corrects it by standardizing procedure. One lists these types of information that can be added, or "augmented", to a demographics database, presumably from databases that vendor provide.

Data quality tools like this one allow definition of rules that integrate into the ETL stream for any data source:

- Matching incoming records with existing data, like identifying to which insured member a claim applies
- Correcting invalid data based on other data in the record, like correcting an out-of-bounds hand-entered measurements based on an independent automated data feed



These are three guiding principles for organizations adding enrichment to their data integration streams:

- Enriched data must be identifiable and audit-able in the target database: Any integration target database should feature complete lineage metadata: where is this data element from, when it was loaded, and what happened to it along the way. This is even more true for data added by interpolating from, augmenting, matching, or correcting source data. Analysts must know which data came directly from the source, which was generated, and the confidence level of the latter.
- Data replaced by enrichment must be available alongside the enriched data: Enrichment processes must store modified or added data in such a way that analysts have access to the "raw" source data. Analysts should be able to independently test enrichment processes and suggest improvements if needed. If, for whatever reason, enrichment doesn't meet specific analysis needs, then they should be able to fall back to the original source data.

**For instance**, I have a subtle elements of one customer with name1 as Name, Pan Card as an Identity evidence, address as an Address furthermore having points of interest of other customer with name2 as a Name, Voterid as an Identity verification and email as an Email ID. At that point After enhancement of information will be consolidated as:

Name1|Name2 as Name, Pancard|Voterid as Identity confirmation, address as Address and email as Email ID.

Customers	Customer 1	Customer 2	Enrichment Process
Name	MN	OP	MN   OP
Identification Method	MN   Pan Card	OP   Voter ID Card	MN Pan Card   OP Voter ID Card
Other Details	MN   Address	OP   Email ID	MN   Address
			OP   Email ID

(Table 4.1: Enrichment Process)

## 4.9 Bureau Run

Bureau Run is the progression after the improvement process. The advanced information is given to agency, they will include their own subtle elements with the improved information.

For Example, one customer has more than one records in distinctive banks. At that point Bureau will discover the subtle elements of one customer relates to their own outcomes and include their own points of interest will be added to enhanced information then at the end of the day improvement procedure is done.

Following Table illustrates the procedure of Bureau Run:

Customer 1	Customer 1	Bureau Run Process
Name: ABC	Name: ABC	Name   ABC
Bank Name: <b>SBI</b>	Bank Name: <b>SBI</b>	ABC   SBI   457812   587412
Account No.: 457812	Account No.: 587412	
Account Type: Saving	Account Type: Current	
Identification: PAN Card	Identification: Voter ID Card	ABC   PAN Card   Voter ID Card
Address Type: Residential	Address Type: Residential	ABC   Residential Address
Email ID: <a href="mailto:abc@gmail.com">abc@gmail.com</a>	Email ID: abc@gmail.com	ABC   <a href="mailto:abc@gmail.com">abc@gmail.com</a>

(Table 4.2: Bureau Run Process)

Customer 1	Customer 1	Bureau Run Process
Name: DEF	Name: DEF	Name   DEF
Bank Name: <b>SBI</b>	Bank Name: <b>PNB</b>	DEF   SBI   123456   PNB   124567
Account No.: 123456	Account No.: 124567	
Identification: PAN Card	Identification: Voter ID Card	DEF   PAN Card   Voter ID Card
Address Type: Residential	Address Type: Residential	DEF   Residential Address
Email ID: <a href="mailto:def@gmail.com">def@gmail.com</a>	Email ID: abc@gmail.com	DEF   <a href="mailto:def@gmail.com">def@gmail.com</a>

(Table 4.3: Bureau Run Process)

#### 4.10 Bureau and ES Enrichment

Bureau and ES Enrichment is the final process of this project. When we run this process then we get the final result and this result is stored in a golden record.

After receiving data from Enrichment Table and Bureau Run Table, a final table will be generated in database from where we apply this final process to get exact clean data.

Consider this below Bureau Run Process Table and Enrichment Table:

Customer 1	Customer 1	Bureau Run Process
Name: ABC	Name: ABC	Name   ABC
Bank Name: <b>SBI</b>	Bank Name: <b>SBI</b>	ABC   SBI   457812   587412
Account No.: 457812	Account No.: 587412	
Account Type: Saving	Account Type: Current	
Identification: PAN Card	Identification: Voter ID Card	ABC   PAN Card   Voter ID Card

Address Type: Residential	Address Type: Residential	ABC   Residential Address
Email ID: <a href="mailto:abc@gmail.com">abc@gmail.com</a>	Email ID: abc@gmail.com	ABC   <a href="mailto:abc@gmail.com">abc@gmail.com</a>

(Table 4.4: Bureau Run Process)

Customer 1	Customer 1	Bureau Run Process
Name: DEF	Name: DEF	Name   DEF
Bank Name: <b>SBI</b>	Bank Name: <b>PNB</b>	DEF   SBI   123456   PNB   124567
Account No.: 123456	Account No.: 124567	
Identification: PAN Card	Identification: Voter ID Card	DEF   PAN Card   Voter ID Card
Address Type: Residential	Address Type: Residential	DEF   Residential Address
Email ID: <a href="mailto:def@gmail.com">def@gmail.com</a>	Email ID: abc@gmail.com	DEF   <a href="mailto:def@gmail.com">def@gmail.com</a>

(Table 4.5: Bureau Run Process)

Customers	Customer 1	Customer 2	Enrichment Process
Name	MN	OP	MN   OP
Identification Method	MN   Pan Card	OP   Voter ID Card	MN Pan Card   OP Voter ID Card
Other Details	MN   Address	OP   Email ID	MN   Address
			OP   Email ID

(Table 4.6: Enrichment Process)

Now, after Bureau and ES Enrichment Process, the record will be stored in a table such as Golden Record Table which can be seen in this below table:

Enrichment Table	Bureau Run Table	Bureau & ES Enrichment Table (Golden Record)
Name   MN   OP	Name   ABC   DEF	MN PAN Card Address
MN Pan Card   OP Voter ID Card	SBI   457812   587412 SBI   123456   PNB   124567	OP Voter ID Card Address Email ID
MN   Address	PAN Card   Voter ID Card	ABC SBI   457812   587412 PAN Card Residential Address <a href="mailto:abc@gmail.com">abc@gmail.com</a>
OP   Email ID	Residential Address	DEF SBI   123456   PNB   124567 Voter ID Card Residential Address <a href="mailto:def@gmail.com">def@gmail.com</a>
	<a href="mailto:abc@gmail.com">abc@gmail.com</a> <a href="mailto:def@gmail.com">def@gmail.com</a>	

(Table. 4.7: Bureau and ES Enrichment Table / Golden Record Table)

## 4.11 Technical Specification

### 4.11.1 Hardware Requirements

Processor	:	Pentium IV
Hard Disk	:	40GB
RAM	:	512MB or more

### 4.11.2 Software Requirements

Operating System	:	Windows
Programming Language	:	Java
Web Applications	:	JDBC, Servlets, JSP
IDE/Workbench	:	My Eclipse 6.0
Database	:	Oracle 10g

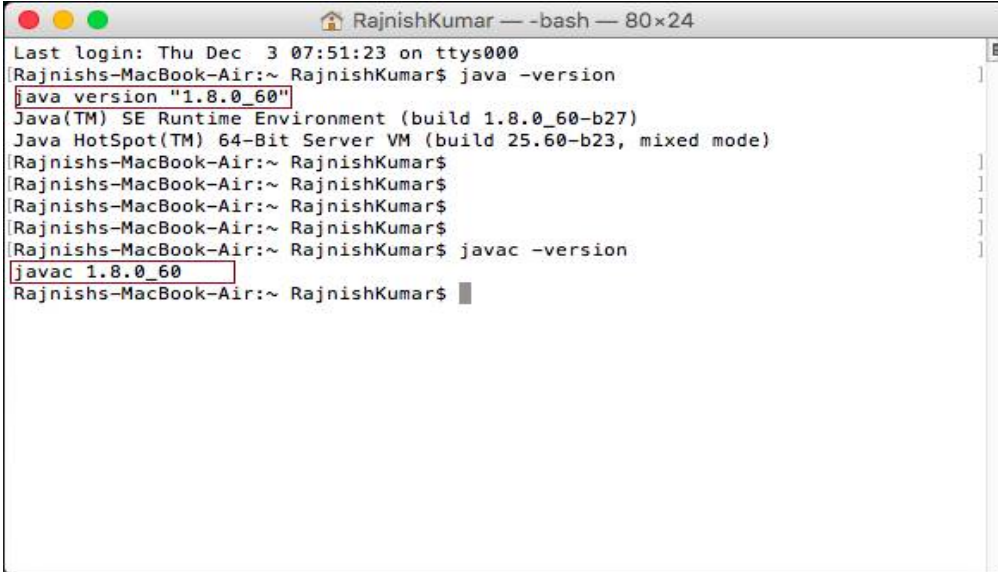
## 4.12 Elastic Search Setup:

❑ **Step 1: Check Java Version**

❑ **Step 2: Download from [www.elasticsearch.co](http://www.elasticsearch.co)**

❑ **Step 3: Installation: \$ bin/./Elasticsearch**

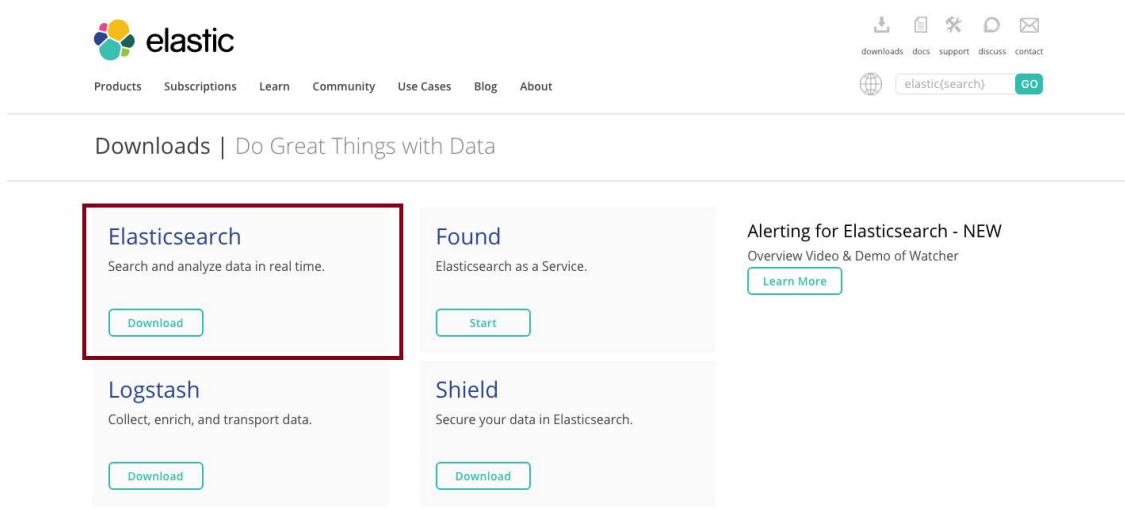
- ❑ Step 1: Check Java Version (Make sure java version should > 1.6)

A terminal window titled 'RajnishKumar — -bash — 80x24'. The output shows the command 'java -version' being executed, resulting in 'java version "1.8.0\_60"'. Below this, it shows 'Java(TM) SE Runtime Environment (build 1.8.0\_60-b27)' and 'Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)'. The prompt then changes to 'RajnishKumar\$'. The command 'javac -version' is then entered, resulting in 'javac 1.8.0\_60'. The prompt changes back to 'RajnishKumar\$'.

```
Last login: Thu Dec 3 07:51:23 on ttys000
Rajnishs-MacBook-Air:~ RajnishKumar$ java -version
java version "1.8.0_60"
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
Rajnishs-MacBook-Air:~ RajnishKumar$
Rajnishs-MacBook-Air:~ RajnishKumar$
Rajnishs-MacBook-Air:~ RajnishKumar$
Rajnishs-MacBook-Air:~ RajnishKumar$
Rajnishs-MacBook-Air:~ RajnishKumar$ javac -version
javac 1.8.0_60
Rajnishs-MacBook-Air:~ RajnishKumar$
```

(Fig. 4.11: Checking Java Version)

- ❑ Step 2: Download from [www.elasticsearch.co](http://www.elasticsearch.co)



(Fig. 4.12: Download Page)

❑ Step 3: Installation: \$ bin/./elasticsearch

```

bin — java -Xms256m -Xmx1g -Djava.awt.headless=true -XX:+UseParNewGC -XX:+UseCo...
Last login: Thu Dec 3 07:52:14 on ttys000
Rajnishs-MacBook-Air:~ RajnishKumar$ cd Documents/Misc/CHM/
Rajnishs-MacBook-Air:CHM RajnishKumar$ cd elasticsearch-1.6.0/
Rajnishs-MacBook-Air:elasticsearch-1.6.0 RajnishKumar$ cd bin
Rajnishs-MacBook-Air:bin RajnishKumar$ ./elasticsearch
[2015-12-03 07:59:45,654] [INFO] [node ] [Jane Foster] version[1.6.0]
, pid[1116], build[cdd3ac4/2015-06-09T13:36:34Z]
[2015-12-03 07:59:45,657] [INFO] [node ] [Jane Foster] initializing .
..
[2015-12-03 07:59:45,670] [INFO] [plugins ] [Jane Foster] loaded [], sit
es [kopf]
[2015-12-03 07:59:45,879] [INFO] [env ] [Jane Foster] using [1] data
paths, mounts [[/ (/dev/disk1)], net usable_space [34.4gb], net total_space [232.6gb],
types [hfs]
[2015-12-03 07:59:53,341] [INFO] [node ] [Jane Foster] initialized
[2015-12-03 07:59:53,341] [INFO] [node ] [Jane Foster] starting ...
[2015-12-03 07:59:53,636] [INFO] [transport ] [Jane Foster] bound_address
{inet[/0:0:0:0:0:0:0:9300]}, publish_address {inet[/192.168.0.25:9300]}
[2015-12-03 07:59:53,696] [INFO] [discovery ] [Jane Foster] elasticsearch/
tnRZgSs7TeGw5vMJsTUb3Q
[2015-12-03 07:59:57,508] [INFO] [cluster.service ] [Jane Foster] new_master [Ja
ne Foster][tnRZgSs7TeGw5vMJsTUb3Q][Rajnishs-MacBook-Air.local][inet[/192.168.0.25:9300]]
, reason: zen-disco-join (elected_as_master)
[2015-12-03 07:59:57,549] [INFO] [http ] [Jane Foster] bound_address
{inet[/0:0:0:0:0:0:0:9200]}, publish_address {inet[/192.168.0.25:9200]}
[2015-12-03 07:59:57,550] [INFO] [node ] [Jane Foster] started
[2015-12-03 07:59:57,606] [INFO] [gateway ] [Jane Foster] recovered [6]
indices into cluster_state

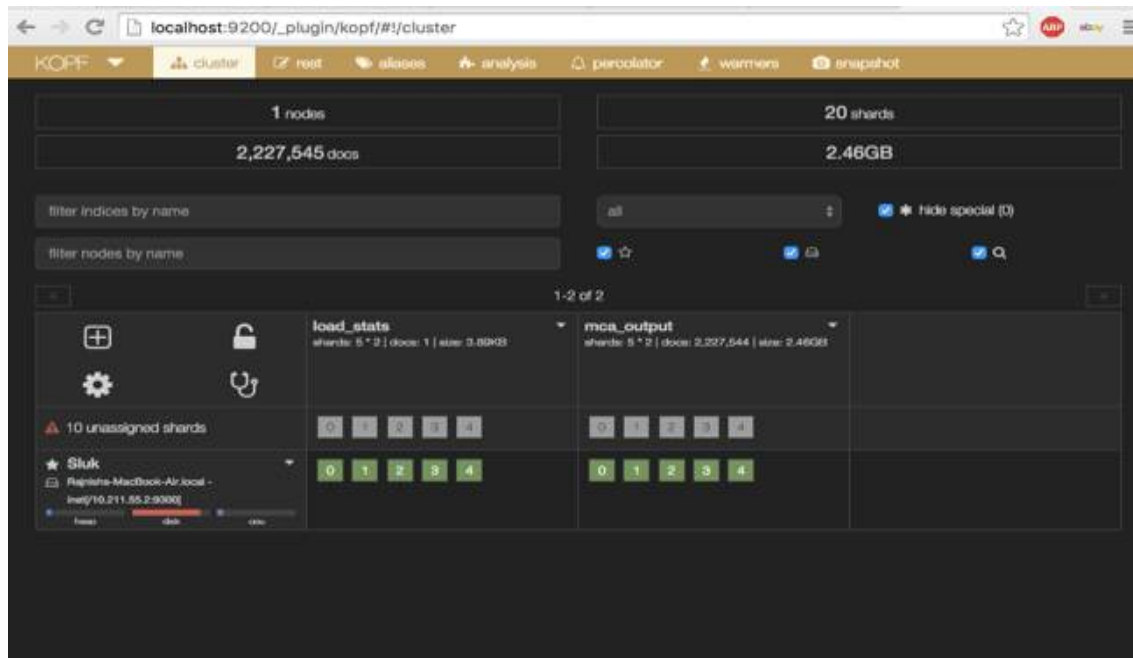
```

(Fig. 4.13: Installation of Elasticsearch)

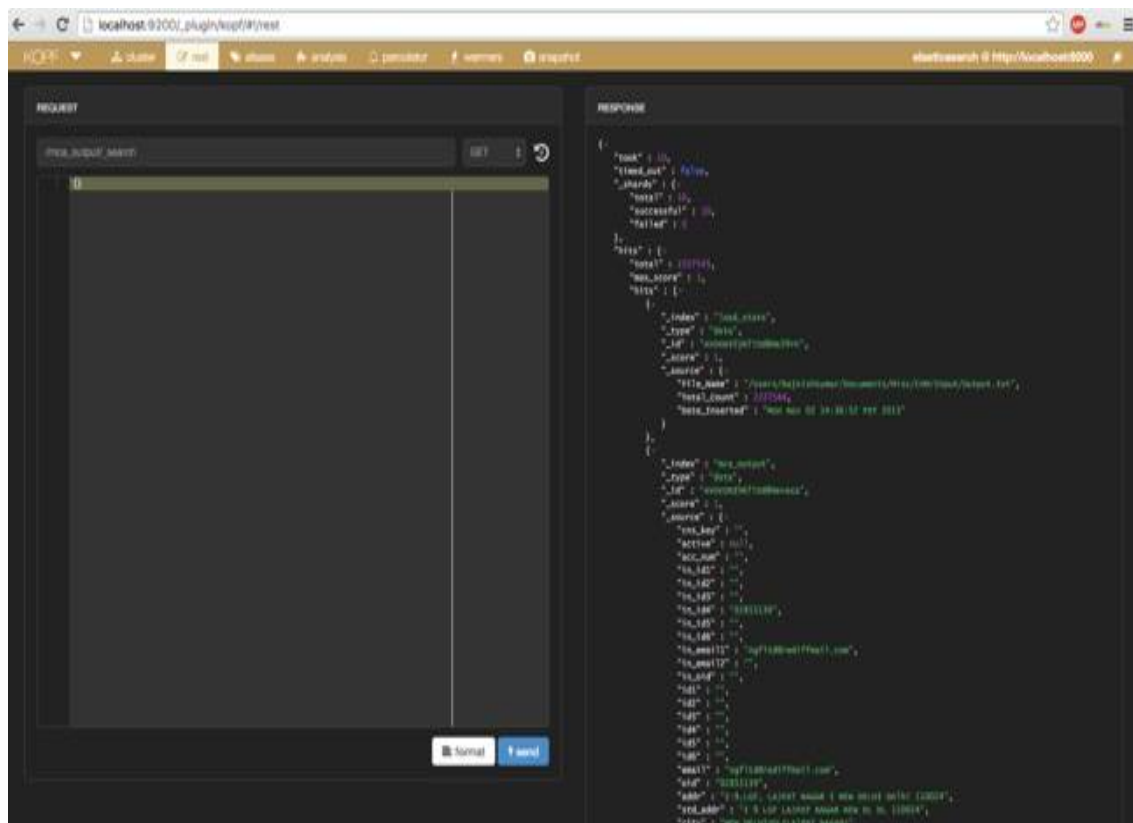
## 4.13 Kopf

- ❑ Kopf is a plugin which is used for elasticsearch.
- ❑ It is generally used for large clusters.
- ❑ It offers filters in Nodes by Name and type, indexes by Name and type.
- ❑ Kopf provides access a lots of elasticsearch API features.
- ❑ It also provides standardization and optimization.
- ❑ It hides special indexes created internally by elasticsearch.
- ❑ It offers a JSON Editor for RESTful services.
- ❑ We can retrieve information from the json object.





(Fig. 4.14: Kopf Loading)



(Fig. 4.15: Kopf Hits)

## 4.14 Sample Codes

### // Building Indexes:

```
XContentBuilder builder;
    IndexResponse response = null;

    builder =
XContentFactory.jsonBuilder().startObject().field("user",
"rajnish").field("postDate", new Date())
                .field("message", "trying out
Elasticsearch").endObject();

    String json = builder.string();

    response = client.prepareIndex("roh", "tweet"/* , i + "
*/).setSource(json).execute().actionGet();
    // }
    // Index name
    String _index = response.getIndex();
    // Type name
    String _type = response.getType();
    // Document ID (generated or not)
    String _id = response.getId();
    /
    long _version = response.getVersion();

    System.out.println(_index + ";" + _type + ";" + _id + ";"
+ _version);

    System.out.println("Time taken " +
(System.currentTimeMillis() - start));

    // on shutdown
    client.close();
}
```

### // Replacement Standardization

```
public static String replaceTermsInCompanyName(String CompanyName) {
```

```

        String arr[] =
StringUtils.splitPreserveAllTokens(CompanyName.toUpperCase(), " ");
        for(int i=0; i<arr.length; i++) {

            if(cleaner.conversionLookUp.containsKey(arr[i])) {

                System.out.println("CompanyName" +
cleaner.conversionLookUp.size());
                arr[i] = cleaner.conversionLookUp.get(arr[i]);
            }
        }

        StringBuilder builder = new StringBuilder();
        for(String s : arr) {
            builder.append(s);
            builder.append(" ");
        }

        return builder.toString();
    }

```

## // Group Standardization

```

    public String findGroup(String CompanyName) {

        String arr[] =
StringUtils.splitPreserveAllTokens(CompanyName, " ");

        ArrayList<String> ar = new ArrayList<String>();
        int count = 0;
        for(int i=0; i<arr.length-1; i++) {

            System.out.print("full company name" + arr[i]);
            if(categoryLookUp.containsKey(arr[i])) {

                System.out.println("segment" + arr[i]);
                if(!ar.contains(categoryLookUp.get(arr[i]))) {

                    ar.add(categoryLookUp.get(arr[i]));
                    count++;
                }
            }
        }
    }

```

## // Name Standardization

```

public List<UniNameMatch> score(List<UniName> inqName, List<UniName>
candName, String inqUnicnstype, String candUnicnstype) {

    int ind = 0;
    NameStrCompare strCompare = new NameStrCompare();
    if (StringUtils.isNotBlank(inqUnicnstype) &&
    StringUtils.isNotBlank(candUnicnstype))
        if (inqUnicnstype.equalsIgnoreCase("BAGIC") &&
    candUnicnstype.equalsIgnoreCase("BAGIC")) {
            ind = 1;
        }
    List<UniNameMatch> result = new
    ArrayList<UniNameMatch>();
    Iterator<UniName> itInq = inqName.iterator();

    while (itInq.hasNext()) {
        UniName inq = itInq.next();
        Iterator<UniName> itCand = candName.iterator();
        while (itCand.hasNext()) {

            UniName cand = itCand.next();
            UniNameMatch res = new UniNameMatch();
            if
    (inq.getCnsType().equalsIgnoreCase(cand.getCnsType())) {

                if (inqName != null && candName != null)

                {

                    boolean flag = false;

                    String initLevel =
    getInitialMatchingLevel(inq.getInitials(), cand.getInitials());

                    String nameLevel =
    getNameMatchingLevel(inq.getStdName(), cand.getStdName(), ind);

                    if
    (nameLevel.equalsIgnoreCase("L1")) {
                        nameLevel =
    strCompare.strCompare(inq.getStdName(), cand.getStdName());
                        if
    (nameLevel.equalsIgnoreCase("L2")) {
                            flag = true;
                        }
                    }

                    String key3Level =
    getNameMatchingLevel(inq.getKey3(), cand.getKey3(), ind);

                    if
    (key3Level.equalsIgnoreCase("L1") && flag) {

```

```

        key3Level = "L2";
    }

    res.setInitialLevel(initLevel);
    res.setKeyLevel(key3Level);
    res.setNameLevel(nameLevel);
    res.setCnsType(inq.getCnsType());
    result.add(res);
    }
    }
    }
    }
    return result;
}

```

## // Phone Standardization

```

public String getLevel(String phoneInquiryKey, String
phoneCandidateKey) {

    if (StringUtils.isBlank(phoneInquiryKey)) {
        return "L0";
    }
    if (StringUtils.isBlank(phoneCandidateKey)) {
        return "L5";
    }

    return levenDistCore(phoneInquiryKey, phoneCandidateKey);
}

public String levenDistCore(String inquiry, String candidate)
{

    if (StringUtils.getLevenshteinDistance(inquiry,
candidate) == 0)
        return "L1";

    if ((StringUtils.getLevenshteinDistance(inquiry,
candidate) > 0)

```

```

        &&
        (StringUtils.getLevenshteinDistance(inquiry, candidate) <=
precision1))
            return "L2";

        if (((StringUtils.getLevenshteinDistance(inquiry,
candidate) > precision1) && (StringUtils
.getLevenshteinDistance(inquiry, candidate) <=
precision2)))
            return "L3";

        return "L4";
    }
}

```

## // Duplicate Standardization

```

public void getStats(Client client, String ind,String bu) {

    SearchResponse q1 = client
        .prepareSearch(ind)
        .setTypes("data")

        .setQuery(QueryBuilders.filteredQuery(QueryBuilders.matchAllQu
ery(), FilterBuilders.termFilter(KeyConstants.UNICNSTYPE, bu)))

        .addAggregation(AggregationBuilders.terms("grp_by_custid")

        .field(KeyConstants.CUSTOMER_ID)

        .order(Order.count(false))

        .size(2))

        .setSize(0)
        .execute()
        .actionGet();

    Terms t1 = q1.getAggregations().get("grp_by_custid");

    Collection<Bucket> b1 = t1.getBuckets();

    int count=0;
    for (Bucket bucket : b1) {

        if (bucket.getDocCount() == 1) {

```

```

        count++;

        System.out.println(bucket.getKeyAsText().toString().toUpperCase() + " | " + bucket.getDocCount());
    }
    }
    System.out.println("No of customer Duplicate:"+count);
}
}

```

### // Delete Standardization

```

public class Delete {
    public void deleteCluster(Client client, Set<String>
idList,String indexName, String typeName) {

        BulkRequestBuilder bulkRequest = client.prepareBulk();
        Iterator<String> itr = idList.iterator();

        while (itr.hasNext()) {

            bulkRequest.add(client.prepareDelete(indexName,
typeName,itr.next()));
        }
        if (idList.size() > 0) {
            BulkResponse bulkResponse =
bulkRequest.execute().actionGet();

            client.admin().indices().prepareRefresh().execute().actionGet(
);

            if (bulkResponse.hasFailures()) {

                System.out.println(bulkResponse.buildFailureMessage());
            }
        }
    }
}

```

### // Cluster Formation after Delete

```

public void afterDeleteCluster(Client client, Set<String>
idList,Set<String> insertId, String indexName, String indexNameHC,
String typeName) {

```

```

        Map<String, String> setCorrospoundingIds = new
HashMap<String, String>();
        Set<String> setDeleteIds = new HashSet<String>();
        Index index = new Index();

        Iterator<String> it = idList.iterator();

        while (it.hasNext()) {

            String id = it.next();
            int size = 1;

            QueryBuilder finalQuery1 =
QueryBuilders.multiMatchQuery(id, KeyConstants.CLUSTER_KEY,
KeyConstants.CLUSTER_KEY_HC).type(null);

            SearchResponse rs = client

                .prepareSearch(indexNameHC)

                    .setTypes(typeName)
                    .setQuery(finalQuery1)
                    .setSize(size)
                    .execute()
                    .actionGet();

            Iterator<SearchHit> itr = rs.getHits().iterator();

            while (itr.hasNext()) {

                SearchHit searchHit = itr.next();
                setDeleteIds.add(searchHit.getId());
                Map<String, Object> temp =
searchHit.sourceAsMap();
                @SuppressWarnings("unchecked")
                List<HashMap<String, String>> list =
(ArrayList<HashMap<String, String>>) temp.get("cands");

                Iterator<HashMap<String, String>> itlist =
list.iterator();

                while (itlist.hasNext()) {

                    Map<String, String> temp1 =
itlist.next();

                    if
(!idList.contains(temp1.get(KeyConstants.CLUSTER_KEY)))

                        setCorrospoundingIds.put(temp1.get(KeyConstants.CLUSTER_KEY),
insertId.iterator().next());

                    if
(!idList.contains(temp1.get(KeyConstants.CLUSTER_KEY_HC)))

                        setCorrospoundingIds.put(temp1.get(KeyConstants.CLUSTER_KEY_HC)
, insertId.iterator().next());}}

```



**// Fetching Candidate Value**

```

public SearchResponse getCandidate(Client client, UniCns cns, String
index) {
    UniID id = cns.getUniId();
    int size = 333;
    String input_id = "";
    if (id != null) {
        if (StringUtils.isNotBlank(id.getID1()))
            input_id = input_id + id.getID1();
        if (StringUtils.isNotBlank(id.getID2()))
            input_id = input_id + " " + id.getID2();
        if (StringUtils.isNotBlank(id.getID3()))
            input_id = input_id + " " + id.getID3();
        if (StringUtils.isNotBlank(id.getID4()))
            input_id = input_id + " " + id.getID4();
        if (StringUtils.isNotBlank(id.getID5()))
            input_id = input_id + " " + id.getID5();
        if (StringUtils.isNotBlank(id.getID6()))
            input_id = input_id + " " + id.getID6();
        if (StringUtils.isNotBlank(id.getOtherID()))
            input_id = input_id + " " + id.getOtherID();
    }
    input_id = input_id.trim();

    String nameKey1 = "";
    String nameKey4 = "";

    Iterator<UniName> it = cns.getUniName().iterator();

    while (it.hasNext()) {
        UniName temp = it.next();
        nameKey1 = nameKey1 + " " +
StringUtils.replace(temp.getKey1(), "^", " ");
        nameKey1 = nameKey1.trim();

        nameKey4 = nameKey4 + " " +
StringUtils.replace(temp.getKey4(), "^", " ");
        nameKey4 = nameKey4.trim();
    }
}

```

**// Loading Index in Cluster**

```

public static void loadStatIndexer(Client client, final String
loadStatIndex, final String process , final String fileName, final

```

```

String insertDate, final int totalCount) {

    try {

        XContentBuilder builder = XContentFactory
            .jsonBuilder()
            .startObject()
            .field("Process",
process)

            .field("File_Name", fileName)

            .field("Total_Count", totalCount)

            .field("Date_Inserted", insertDate)

            .endObject();

        String json = builder.string();

        System.out.println(json);

        client.prepareIndex(loadStatIndex,
"data").setSource(json).execute();

        builder.close();
    } catch (Exception stg) {
        stg.printStackTrace();
    }
}

```

## **Conclusion and Future Scope**

## Conclusion

We have used very successful procedure with the help of Elasticsearch so can conclude that user will feel free to get cleaned data. We can be very helpful to provide exact data in banking sector, financial institution, and information bureau such as in PAN Card Centre, Voter ID Card Organization, Card Organization such as Master Card, Visa Card and in many more.

Generally, when a user request our service then we use these five process:

1. Data Loading
2. Inquiry
3. Enrichment
4. Bureau Run
5. Bureau and ES Enrichment

All these above five process are done through Elasticsearch.

If any user request our service then they will be able to take benefit for:

- Better Performance
- Enhanced Search Result
- Reduced Server Load

## Future Scope

These are the future works:

- Standardize the retrieved data
- Optimization of informational data
- Fetching Data into indexes