The Corporate-Political Nexus

A thesis submitted in fulfillment
of the requirements for

M.TECH MAJOR PROJECT
by

AMARTYA CHAUDHURI

Entry No. 2014MCS2117

Under the guidance of

Dr. AADITESHWAR SETH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
INDIAN INSTITUTE OF TECHNOLOGY DELHI.

JUNE 2016.

Certificate

This is to certify that the thesis titled **The Corporate-Political Nexus** being submitted by **AMARTYA CHAUDHURI** for the award of **Master of Technology** in **Computer Science & Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Computer Science & Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Dr. AADITESHWAR SETH
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Abstract

In this project we try to facilitate the study of distribution of power across and within various Indian power institutions by analyzing their inter-linkages, family trees, time-lines, transactions, etc. to make more sense of the big political and corporate handshakes. In that direction, we envision to construct a system to collect data in this regard from various sources and integrate all of it into a single data store for others to use.

Our aim is to disseminate these findings to the mass society using a crowd-sourced system, and use mass language for such a flow of information, and in this way, enhance governmental accountability and transparency using the notion of Open Data and crowd-sourcing.

Acknowledgments

I would like to express my heartiest gratitude to my supervisor Dr. Aaditeshwar Seth for guiding this work with utmost interest and scientific rigor. I thank him for setting high standards, giving me freedom to explore multiple facets of the problem and teaching me the value of analytical thinking and hard work. I am also grateful to Manoj Kumar, Anirban Sen and Dipanjan Chakraborty who have helped a great deal by providing their support during difficult times and whose suggestions went a long way in making this work a reality. I am highly indebted to Preeti Rani, Mridul Goel and Md. Imran for actually sitting with me and working on perfecting the final system. I would also like to thank my family and friends for their love and support and bearing with me for my constant absence.

AMARTYA CHAUDHURI

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Democracy is a system of government by all the eligible members of a state. In the words of Abraham Lincoln, it is *the government of the people, by the people, for the people.*

But from time to time, it has been observed that a democratic government can be loosely described in terms of a collection of *hegemonies* and *counter hegemonies*. By a *hegemony*, we actually mean the dominance over a society by people controlling important resources in the nation (natural or artificial) namely military, politics, academics, businesses, media etc. The distribution of power in hegemonic parties are often hereditary (among close ties in family) instead of elected representatives. Interactions among the persons of these important fields are also common for their effective functioning (or often to safeguard self-interests).

Yet the beauty of this system lies in the concept of *counter hegemony* - a part of the civil society which functions to keep check on the hegemonic bodies. All the oppositions and protests against government and/or other powers, and such kind of information dispersion, comes under the flag of counter hegemony. To summarize, a democratic society can be said to controlled and influenced by these opposing forces of hegemony and counter-hegemony.

In this regard, accountability and transparency in government are the key requirements in order to obtain an ideal democratic society. Unfortunately the lack of proper knowledge about the *power-houses* has led to new forms of *collective* dictatorship where public rights and voices are sometimes rendered ineffective.

To account for this growing problem of opacity we have tried to study the social network of Indian politicians and corporates and disseminate the information to the common public through our present work.

1.1 Objective

The intent has been to complete the following tasks:

- To **collect data from semantic web** (and other sources) to form a database (henceforth referred to as **"knowledge base"** / **"knowledge graph"**)
- To construct a **highly integrated, structured, error-free data store** by collecting and integrating political, corporate, bureaucratic and other kind of similar data which otherwise is scattered at random, dis-connected sources on the web.
- To provide **a data mash-up from different fields** to further help the academicians, journalists, data-enthusiasts etc.
- To **monitor the top players in Indian society** - mainly in the spheres of politics and business in India.
- To **disseminate this information to public** to bring about accountability and transparency.
- To seek answers to questions like -
  - *Who were the big players in politics and business in the Indian domain?*
  - *Is there any influence (or possibility of it) of political field by a person in corporate field or vice-versa?*
  - *How important is one politician in a network of politicians (or a businessperson in a business network)?*
  - *Where does the actual power reside in a democracy?*

We believe that through our work, we will be able to show how such system of inter-disciplinary data helps to find interesting patterns and spread more knowledge.

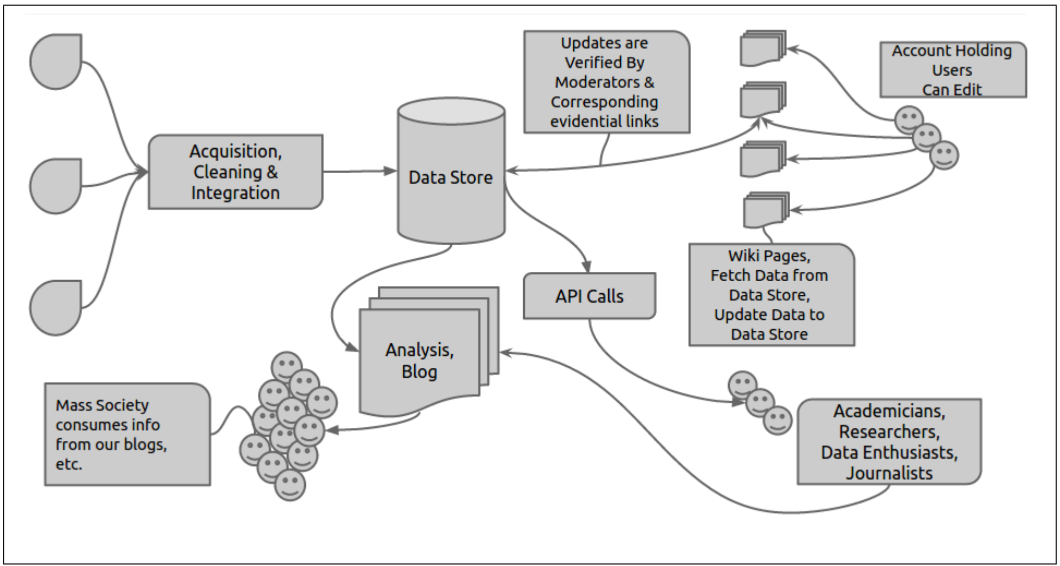


Figure 1.1: Big picture of the system envisioned

1.2 Motivation & Related Work

In his book **The Power Elite** [28], C. Wright Mills calls attention to the interwoven interests of the leaders of the military, corporate, and political elements of society and suggests that the ordinary citizen is a relatively powerless subject of manipulation by those entities. His book deals with the power elite in US. But the hierarchy he proposes is more or less the same across all countries. Power rests with the top one percent in an economy. We plan to create a watchdog for that one percent. One interesting list to accompany this direction could be the Forbes list [16] of 147 companies that control everything.

French economist Thomas Piketty in his famous work **Capital in the Twenty-First Century** [30] focuses on wealth and income inequality in Europe and the United States since the beginning of the industrial revolution. He proposes a global system of progressive wealth taxes to help reduce inequality and avoid the vast majority of wealth coming under the control of a tiny minority. We plan to collect, integrate, visualize and open such data for Indian terrain to let data fanatics carry out such works to understand this inequality.

British writer and historian Patrick French, in his book **India: A Potrait** [24] has stated many such interesting patterns in Indian politics where he argues that almost all of the young Indian politicians in the Indian Parliament are hereditary. In fact, patterns similar to this can be seen over the entire political Indian scene. One can find interesting overlaps, family ties, social links within these power houses. In a survey, **Who owns your media?** [20], we find that even the media is an entity of importance in the game of power and most of the famous and powerful politicians and businessmen try to pull strings in this domain. Another interesting case is Jayant Sinha's family tree [22] and their business holdings. As of 2015, he was the Minister of State for Finance and a Member of Indian Parliament and had links to lot of powerful companies.

Research along the area have been prominent across countries. **Sastry** [31] shows how crime and money play important role in Indian elections. In a related work **Vaishnav** [33] explain why do Indian parties elect criminal candidates and why they win. **Kapur** [25] connects the hidden relationships between politicians and builders. He argues that where elections are costly but accountability mechanisms are weak, politicians often turn to private firms for illicit election finance and that where firms are highly regulated, politicians can exchange policy discretion or regulatory forbearance for bribes and monetary transfers from firms.

Works like what we propose have already been done for countries like USA, UK, Chile etc. We have examples like **LittleSis** [17], **Poderopedia** [21] where journalists, developers, analysts come together to put up profiles of important entities, institutions of the society and highlighted the connections between them. Littlesis (opposite of Big brother) in one hand exists in USA from the political and economical data available there. Poderopedia is a similar site in Chile. These sites feature separate pages of people in power in USA, their connections to different institutions and other entities , work history, visualizations of the connections to educate masses etc. Other than producing awareness to people about the corporate- political connections, these sites also allow public to register and collaborate in data entry processes and have an API system to promote further use of their data for research purposes.

Such system in absence of digital data/ structured data and other human factors is difficult in India. But various local and national initiatives have been started. **Association for Democratic Reforms** [15] for example has sites like **MyNeta** [19] to disseminate information about political leaders of India.

Our vision is to produce a system similar in lines to the websites embedded with the power to query interesting connections, find interesting visualizations, and help raise suspicious issues.

The core two things that required our special attention when working with several data sources is the process of **modeling the database as a graph** and method of **resolving same entities from different data sources** . These two things are problems with extensive study of their own.

Entity resolution has been studied since 1946 by works of *Halber.L.Dunn* [23]. Basic method is to match a pair of strings accurately to determine possible similar entities. Since then, several string matching algorithms has been used for this purpose. The **levenshtein algorithm** [27] gives score based on no of edits to convert one string to another. It is especially useful to deal with the problem of mis-spelt records. Improving on that the **Jaro-Winkler** [34] looks at matching characters within a small range while giving scores. This is suitable for short strings such as names and fits our purpose well. Another class of string matching algorithms looks at phonetics to resolve entities with similar sounding text. The **Soundex algorithm** [26] is one of the most well known. These algorithms use English language pronunciations to create an index of the text. For our purpose, we have used the Jaro-Winkler and a variation of Soundex **(Double Metaphone)** [29].

The graph model of the data is required since we are emphasizing the relationships among the data. Graph databases as a concept existed long since mid-1960s when **IBM IMS** [32] [8] supported graph structures in its hierarchical model. Graph databases allow one to give semantic queries over data relationships. A graph database models the entities as nodes and relationships among them as edges between nodes. Internally, a graph database stores the data as a relational table (**MariaDB** [9]) or through document-value stores (**Neo4j, OrientDB**[11][12]). For the purpose of our project we have used the Neo4j database to store the core-data.

1.3 Thesis Overview

The rest of the work has been divided into following sections:

1. **Constructing the Social Network** - The choice of graph database has been explained here, with a detailed description of system's data model. The challenges and methods adopted while collecting and integrating data too have been described. Entity Resolution, which forms the basis of the entire system and which is one of the most important challenges of the work, has been discussed at length.

2. **Design of Power Elites Web App** - A complete overview of the system is provided with all the possible actors. All the components of the system, their functionalities and interactions with the actors of the system have been explained here. All the internal details of the wiki and how data is actually gathered and resolved is discussed thoroughly.

3. **Visualizations and Analysis**- The results obtained from the constructed system have been described here. Interesting connections among and within the power houses have been listed here along-with the resultant observations.

4. **Conclusion and Further Work** - Various ways the present work can be expanded are listed here along with the conclusion of the work in hand. All important future additions and improvements have been documented.

Chapter 2

Constructing the Social network

The driving fuel for any social network is the data it represents. The problem in constructing a linked-data system such as this one is always the data and the entropy it brings. The challenges are always the usual ones - unavailability of data, noise in the collected data, no authentic source, and many-a-times no structure in the data. Even if we are successful in collecting and cleaning the linked data we want, the way we go about integrating all this variety of information in a single data-store is itself an another challenge.

The choice of data-store matters here the most because one would like to query the data and unearth interesting relations between parties111cipating entities. Henceforth, a person or an *organization* will be called an *entity*. These entities would be linked to each other by certain edges/links just like in a graph which we will normally call *relations*. All these challenges are elevated manifold when one wishes to resolve entities from different datasets into a single unified entity.

Here we describe in order our choice of data storage, our core data model for the linked data, data collection practices and data sources, data integration methodology, and finally the necessary evil in such a system - entity resolution.

2.1 Everything is a graph

We begin by describing how we are going to actually store any kind of linked data we get and why our approach is a sensible one. We describe here where our core data is stored and how our core data model looks like. It has to be stated at the onset that care has been taken to ensure that whatever data goes in our core data modal is non-redundant, free of noise and verified. As already stated, our goal is to construct a social network between politicians, companies, entrepreneurs, military personnel, bureaucrats, political parties, universities, movie actors, and other important entities in the usual power hierarchy. Also, we want to be able to model all kinds of relationships that can exist between these entities with ease: family-links, donation-links, director-links, ownership-links, subsidiary-links, etc.

Practically any connected data can be represented by a graph. We live in a connected world. There are no isolated pieces of information, but rich, connected domains all around us. Thus, it makes sense to model our core data as a large inter-connected graph where *every node is a real-life entity and every edge represents a real-life link between two of them.*

A graph is composed of two elements: a node and a relationship. Each node represents an entity (a person, place, thing, category or other piece of data), and each relationship represents how two nodes are associated. This general-purpose structure allows you to model all kinds of scenarios - from a system of roads, to a network of devices, to a population's medical history or anything else defined by relationships.

Before beginning to describe how we achieve the above, it is important to understand that even traditional SQL tables are a connected piece of information, and can be modeled using a graph.

What we provide is therefore a graph to the user where he fits any relevant connected data he has. The choice of the data model is clear, but two problems remain - how we are going to store our graph's interconnected data, and how do we query it efficiently for digging out interesting relationships. Our next two sections discuss the same.

2.1.1 Neo4j

Neo4j [11] is our choice of data storage. It is one of the leading JVM based NoSQL graph databases. We build our knowledge base on it as a graph and thus Neo4j is at the center of our entire system. We have found *Neo4j* to be a non-separable asset for our use cases because of the following reasons [14].

1. Only a database that embraces relationships as a core aspect of its data model is able to store, process, and query connections efficiently. While other databases compute relationships expensively at query time, a graph database stores connections as first-class citizens, readily available for any join-like navigation operation. Accessing those already persistent connections is an efficient, constant-time operation and allows you to quickly traverse millions of connections per second per core.

2. Graph databases are designed to mimic the most natural way we tend to model data - the same way you would map it all out on a whiteboard. Your collection of circles, boxes, lines and arrows is - in essence - already a graph.

3. In Neo4j, everything is stored in form of either an edge, a node or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labeled. Labels can be used to narrow searches. Neo4j is very easy to learn and adapt. It's object property model is very intuitive and anything can be modeled on a white board in the form of nodes and edges.

4. Constant time traversals for relationships in the graph both in depth and in breadth due to efficient representation of nodes and relationships.

5. All relationships in Neo4j are equally important and fast, making it possible to materialize and use new relationships later on to shortcut and speed up the domain data when new needs arise.

6. Compact storage and memory caching for graphs, resulting in efficient scale-up and billions of nodes in one database on moderate hardware.

7. It's NoSQL help us in modeling the varied data from different sources we have collected.

8. The cypher query language provided by Neo4j helps in querying the connected data very easily and is very powerful. Also, the Neo4j browser client provided by Neo4j is very good for visualizing the results of the cypher queries.

That said, it is important to note that Google uses Cayley - an open source graph database - to power it's google's knowledge graph.[7]

2.1.2 Property Graph Model Explained

Let us dive into some examples which explain how me model our core data in Neo4j using it's property graph model.[14]

1. The property graph contains connected entities (the nodes) which can hold any number of attributes (key-value-pairs). What this means for us is: a person node can have different attributes his date-of-birth, address, email, sex, etc.

2. Nodes can be tagged with labels representing their different roles in our domain. That said, this unique thing about Neo4j helps us in specifying IS-A relationships via multiple labels on a single Neo4j node. Thus, a node can be labeled as a person, politician, businessman at the same time. The same information is very hard to model in traditional SQL databases.

3. In addition to contextualizing node and relationship properties, labels may also serve to attach meta-data index or constraint information to certain nodes.

4. Relationships provide directed, named semantically relevant connections between two node-entities. A relationship always has a direction, a type, a start node, and an end node. This model is exactly the same as directed edges in a traditional graph structure. Although they are directed, relationships can always be navigated regardless of direction using cypher query in Neo4j.

5. A relationship can also be labeled by a single label specifying what kind of a relationship exists between two nodes. Thus, if a politician is son-of a businessman we can connect the two nodes by an edge attributed with such a label.

6. Just like a node, a relationship can also hold any number of attributes (key-value pairs). In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths.

7. As relationships are stored efficiently, two nodes can share any number or type of relationships without sacrificing performance.

An example could be the Figure 2.1

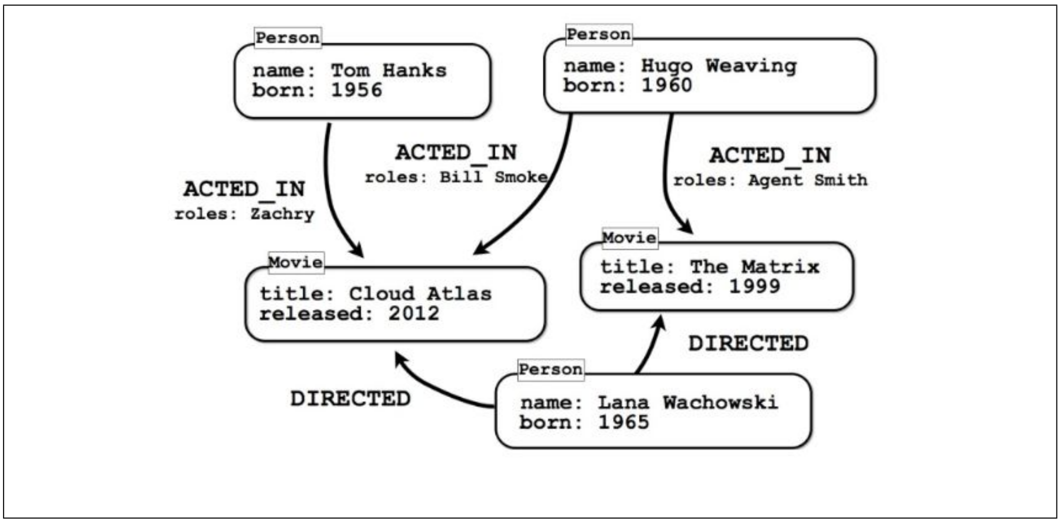


Figure 2.1: An example of property graph model

2.1.3 Cypher

Cypher is a declarative graph query language for the graph database Neo4j that allows for expressive and efficient querying and updating of the graph store. Cypher is a relatively simple but still very powerful language. Very complicated database queries can easily be expressed through Cypher. This allows users to focus on their domain instead of getting lost in database access.

A node is represented like this:
*(:person:politician {name:***'Narendra Modi'***,sex:'M'})*

A relation is represented like this:
*(startnode)-[:relatedto {kind:***'childof'***}]->(endnode)*

The structure here is self explanatory and can be further explored by reading Neo4j manual. Figure 2.2 explains better.

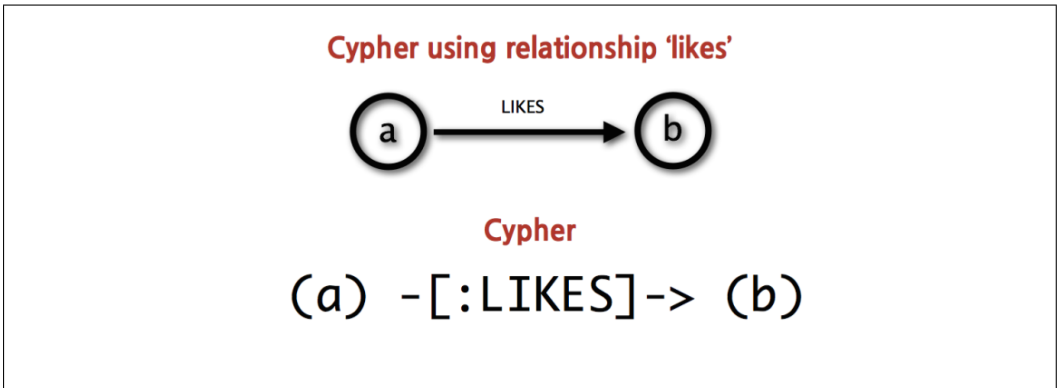


Figure 2.2: Cypher Example

Querying in cypher is as easy as thinking about how you traverse a graph. Here is a simple query to return all the relationships that start from or end at node Naveen Jindal of type politician.

MATCH *(jindal:politician {name:***'Naveen Jindal'***})-[anyrelation]-(anynode)*
RETURN *anyrelation*

Interested readers can redirect here to read more about cypher in Neo4j's manual[10]:

2.1.4 Our restricted property graph model

We had to specify some ground rules to model our data for imposing uniformity on varied data that is being fed into the system. Thus, our property data model follows the rules stated below.

1. As allowed by Neo4j - a node can have more than one label, a relation always has only one label.
2. Every node and a relation has a unique uuid/relid.
3. All nodes are labeled *entity* by default. To make anonymous queries on nodes easier.
4. All living or dead people are labeled *person*.
5. Any person connected to a company as a director/owner is labeled *businessperson*.
6. All operating units are labeled as *organization*.
7. All companies are labeled as *company*.
8. A political party is labeled as both *organization* and *company* alongside *entity*.
9. Other self-explanatory labels for nodes in current core data are: city, state, constituency.
10. Every *entity* will have to have a *name* property. An *aliases* property - a neo4j array - helps in keeping track of different names of an entity.
11. An *entity* can have any number of properties, but these properties if present are validated: *startdate*(int), *enddate*(int), *iscurrent*(boolean). For a person a *startdate* represents his date-of-birth, for a company *startdate* represents it's incorporation-date. The *iscurrent* property

can help us in tracking if the *person* is dead or the *company* is not active.

12. *startdate* and *enddate* are timestamps since epoch - so any date-of-birth or company's incorporation-date will have to be converted to a particular format before pushing to the system.

13. All relationships have to have a property *bidirectional* - to explicitly specify if the relationship goes both ways. This had to be done as Neo4j edges are always directed, though they can be queried without directions.

14. Some labels for relationships in current core data are: *relatedto*, *worksin*, *geoBelongs*.

15. An entity can have any number of properties, but these properties if present are validated: *startdate*(int), *enddate*(int), *iscurrent*(boolean).

16. Any other meta-info will be stored in a separate SQL database that will help in mapping any data point change to it's source and the user who allowed that change. This is better explained in the provenance section 3.5.

We list some images here that better describe the core data model.

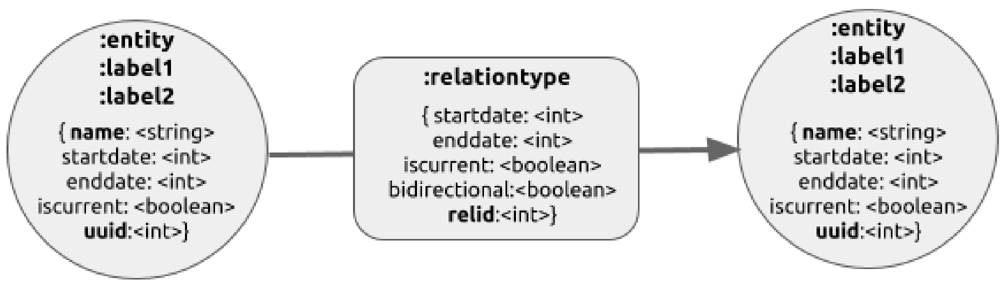


Figure 2.3: Generic model - two entities with a relation

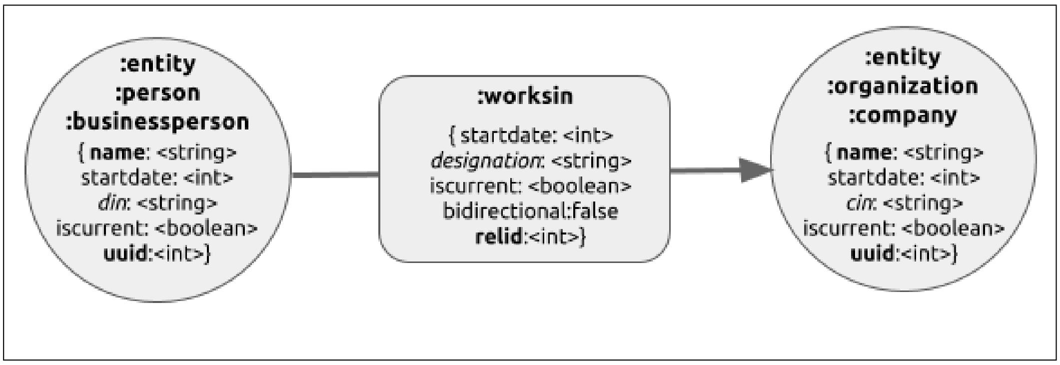


Figure 2.4: Model showing businessperson-organization relationship

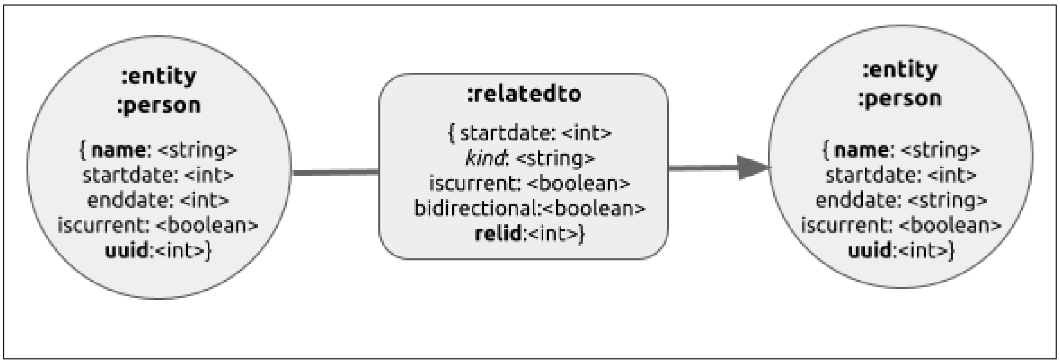


Figure 2.5: Model showing person-person relationship

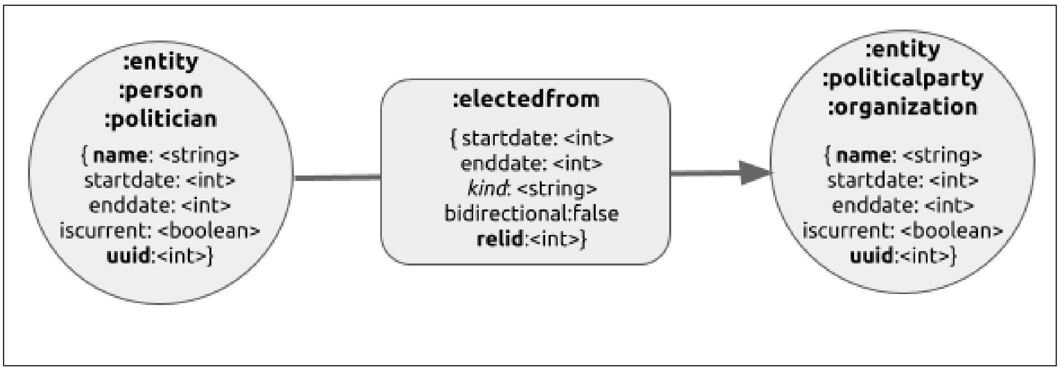


Figure 2.6: Model showing politician - party relationship

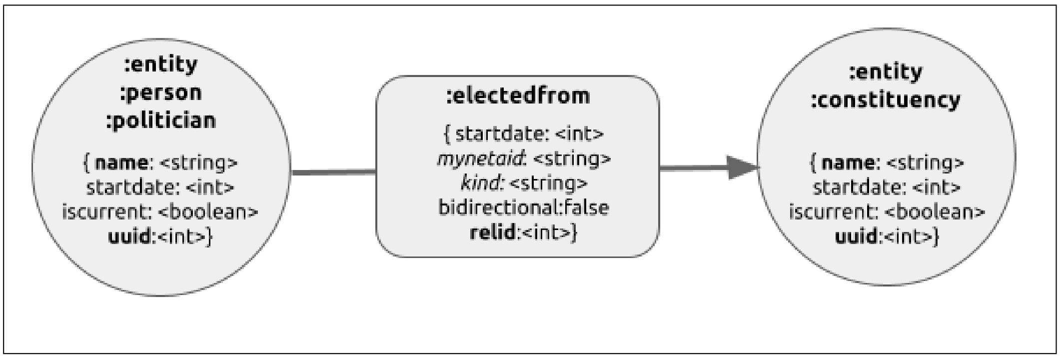


Figure 2.7: Model showing politician - constituency relationship

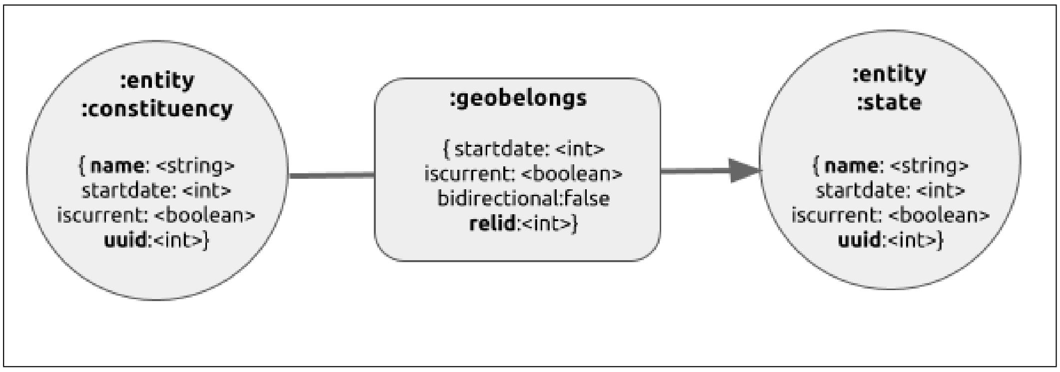


Figure 2.8: Model showing relation between person and his home state

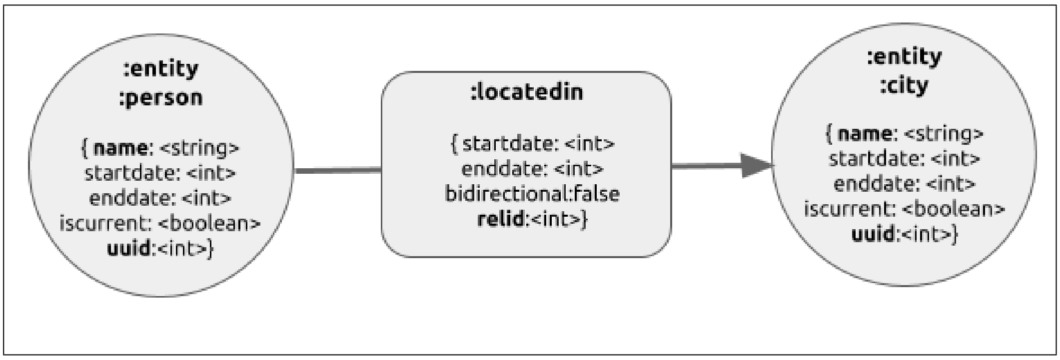


Figure 2.9: Model showing relation between person and his home city

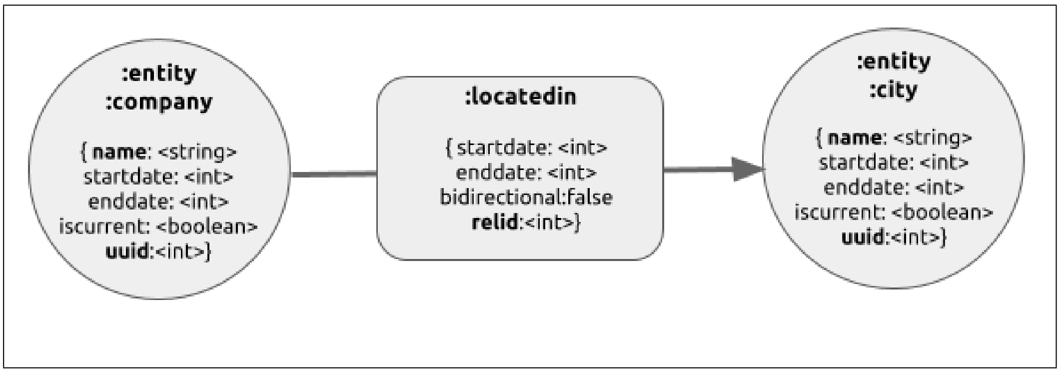


Figure 2.10: Model showing relation between *organization* and its home city

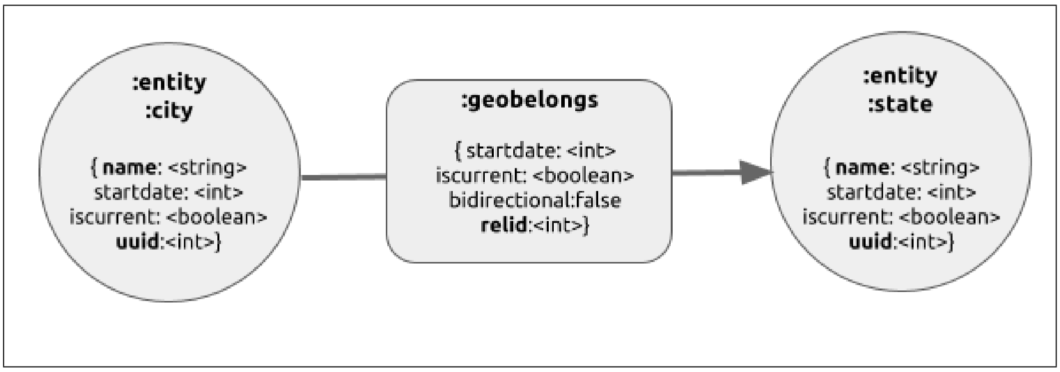


Figure 2.11: Model showing city-state relationship

2.2 Data Collection & Description

Since the aim was to to build a social network for all the Indian power houses, we needed to identify and collect as much varied data as possible. It is only when different kind of data mingle with each other in the system that we can hope to see some hidden patterns or dig out interesting insights. All the visualizations that we obtained from this data are shown in the analysis Chapter 4.

We needed to search for alternate data sources that might help the ongoing work. We have looked for datasets spanning company details, board of directors, Lok-Sabha MPs, subsidiaries, banks, and any other relevant political-corporate data.

Any data that is crawled is not fed into the system directly. Instead, a

REST POST API provided by our system is used to push it to a separate crawl data-store (rather than the core data-store) for moderation first. The data integration REST API is described in 2.3.

2.2.1 Challenges

1. The biggest challenge in data collection is that data for Indian context is not easily available. Open data initiatives like data.gov.in have initiated hope for data enthusiasts but there is still a long way to go.

2. Since data has to be integrated from variety of source points, the credibility of a data item listed can be established strongly but with the added evil of Entity Resolution - which has been explained in section 2.4.

3. Mostly, no unique identifiers for entities appear in datasets that are publicly available. There is a possibility of duplication of data, this is where the ER work comes in handy.

4. Most data contained noise, or erroneous details at times. This sometimes is intentional on the data entry operator's part, sometimes it's just a naive mistake. For example: name for some people was wrongly spelt in some publicly available websites.

5. Data was missing for some datasets. For example, in the affidavit that the politicians have to fill before elections, or in the data collected from companywiki.

6. Names are the biggest problems in such a work. People use titles like Mr., Shri, Shriman, Late, Lt., Mrs., Kumari, Kumar which again add more complexity to the problem.

7. There is no standard data format for dates or addresses even across government departments.

8. Most of the government sites maintain significant data on-line but hidden in a complex web of links and most of the times the data is available in PDF files.

2.2.2 Tools and Methods

We describe here tools and methodology adopted along the way for data collection and cleaning. It is important to note that no single tool can be a cure for all the data-sources. In fact, what we have learned in due course of time is that different kind of data-sources require different approach. The data being noisy due to entry errors and the schema/model/format differences of the sources cause various consistency issues to creep up when compiling them into a single model. There are other issues of not obtaining enough data to churn out any useful information. In that case we have adopted techniques of generating more data from that which is available.

1. A **naive method** that we experimented during the initial work is to fetch an html page through python's urllib library, and then parsing the same with BeautifulSoup library. Advantages of Beautiful soup is that it allows users to get html elements by text query instead of div ids like xpath or css selectors. On the downside, this approach requires all cases of failures handled explicitly in the code. Even the data to be stored is to be explicitly written in desired type of file (csv in our case).

2. **Selenium** - It is basically a web testing framework, mainly used to automate human interactions with an website. Running Selenium is similar to having a human agent using a browser to select items and extract data for you. Since it works as an agent with a browser, many issues get automatically averted like - authentication issues against a bot, easy interactions with browser GUI elements (for eg. - modal panes) etc. Selenium has been extensively used in scraping of the companywiki website for our dataset.

3. **Scrapy** - A framework to create a web crawler bot which can be used to crawl the web and scrape pages. It has almost everything required to crawl/scrape data with ease. It provides a request wrapper to send requests at ease, callback functions to handle errors, ORMs to store the data with ease with whatever format user wants, file down-loader etc. The framework scrapy also uses *twisted*, an asynchronous network library which is event driven. So unlike other tools above, data collection

with scrapy is very fast. The disadvantage however is that of a high learning curve which can be a waste of time if amount of data to be collected is not that high. Scrapy also cannot interact with same page Javascript requests and interact with browser elements. The websites of Loksabha, RajyaSabha for our datasets were scraped using scrapy.

4. **Manual data collection** - When the dataset is small, it is manually collected by using some tool (Google sheet's *ImportHtml* function for example) or by simply doing data entry by hand. The resultant data is free of errors with some trade-off for time. Manual work is also required when ***bootstrapping some dataset*** . Bootstrapping is the process of expanding a dataset using some seed dataset. For our case, we collected data for top 1000 companies by value from Capitaline. From this initial seed, company id no. (CIN) was collected from companywiki site, with their director information. Finally director's pages were looked after to collect information about other companies shared by them. In the end, this data generated a total of 60000 company records and 90000 director records.

5. The **data collected is preprocessed** using the python pandas library. Here issues like encoding mismatch (*utf-8 vs ascii*), text format conversions (like *converting a date given in string to UNIX timestamp format*) and text normalizations (like *name field reduction to FirstName <single space>LastName*) are handled .

2.2.3 Corporate data

The challenges in collecting any corporate data are:

1. Getting government provided unique identification numbers for the company.
2. Getting government provided director identification numbers for their directors.
3. Linking these two IDS.

4. Getting the list of all important companies (with their IDs) over the time.

5. Getting the list of subsidiaries for each company/group. (The toughest job)

We have been able to tackle the first four challenges to a commendable extent but the last one. We first describe the sources we have encountered, and finally how we collected our corpus of 90000 directors and 60000 companies.

2.2.4 Ministry of Corporate Affairs (MCA)

Being the official government source, MCA [18] is the most authentic data source we have found for obtaining the list of companies and LLPs (Limited Liability Partnerships) operating in India. It gives us all the companies registered year-wise before and after 1980 till date in the form of PDFs with their CIN/LLPIN (Corporate Identity Number/Limited Liability Partnership Identification Number) which is unique to a company/LLP. The only downside as always with any PDF data is a lengthy parsing job.

MCA, with its search engine, contains a huge amount of data. Armed with the CIN, we can obtain a lot more information for a company from the MCA site itself. We get the type of the company, the category of the company, it's headquarters, its market capital, date of incorporation and the charges it is facing. Not only that, we also obtain the signatories of a company from the MCA site as well i.e. their board-of-directors, with their unique DIN(Director Identification Number). This can again be useful in entity resolution.

But MCA with all its data poses a lot of problems to deal with:

1. MCA has an exhaustive list of all legally registered companies which is a really big number to process. As such, not only crawling them will be a lengthy task, but also processing the data after crawled and filtering out the relevant companies of interest.

2. Crawling MCA and scraping it is not a very straight forward task as the web UI within it contains a bad control flow with manipulation of many

browser GUI elements which as of now can be performed by testing frameworks like Selenium only. This is a real performance bottleneck determined by speed of Selenium and MCA server.

3. Although it allows one to look for director DINs and other information from a CIN, the reverse is not implemented for MCA website. The site does not also provide enough filtering options to filter out data in first place.

2.2.5 Initial approach, Capitaline database and CompanyWiki

Initially we tried to collect company data by searching the names of politicians as potential directors. This helped us to find companies associated with politicians as our proof-of-concept for potential overlap between coporate-politcians.

But the problem was that most of the companies that the politicians are linked to are small businesses for which we have little interest. Thus, we changed our approach to obtain the companies of interest (most valued companies for example), and find the entities linked with them. These entities of the top companies are actual power elites by our definition. Any links between them and politicians (direct or indirect) are our actual point of interests. At this juncture, we utilized the data from **Capitaline** [4]. Capitaline is a database containing info on Indian companies.

The data collected consisted of following information-

- Company name
- Director
- Subsidiaries, if any
- Values in crores
- Donations to political parties

However, the problem was that Capitaline had no unique ids like CIN, DIN to uniquely determine a company or director and help us in resolution. So we decided to generate a new dataset of our own. Using the Company Name and Value field we picked top 1000 valued company from Capitaline and looked for relevant information in other data-stores like CompanyWiki.

CompanyWiki is another website containing data from mostly MCA. It helped us by giving a better platform with more options of searching/ querying the data. It also contained government IDs for the entries (CIN for companies, DIN for directors etc.) which helped us to link entities. The website also allowed us to search on CINs, DINS / get CINs given DINs and vice-versa.

For our purpose we obtained our director sharing data by following the under-mentioned steps.

1. Obtain top 1000 valued company from Capitaline data - an SQL query on database table storing the company name and value ordered by value and limit to 1000 entries.
2. Manual annotation of 1000 companies to get its CIN. This was done by searching for the company name in CompanyWiki and manually verify for exact same entities.
3. Running Selenium script for taking the list of IDs to get all director names, DINs from CompanyWiki.
4. Using these to get all the companies, CINs where the corresponding director is a member. Here also Selenium was used to get the company names, CINs from the list of DINs inputted. The script took entire 3 days to complete the task. Advantage of using the CINs and DINs here became useful when duplicates are filtered with the help of these for both companies and directors.

Thus in this way a dataset of 1000 companies and 9400 directors was expanded to a dataset of 90000 directors and 60000 companies. This data formed the starting point of our Core Database.

2.2.6 Political data

Our task in this domain was to find personal information of MPs and their dependents. So we went looking in Wikipedia for various political members and their families. But Wikipedia is highly unstructured and crawling, parsing Wikipedia even with professional extractors is a tedious & challenging task.

We started with these problem statements in our mind:

- How to get info on all 16 Lok Sabha's MPs?
- How to get the family information of MPs?

MyNeta

MyNeta is a site launched by Association of Democratic Reforms (ADR) to provide public awareness about Indian politicians. They contain several important information about Indian MPs including -

- Personal information
- State, Constituency of MP
- Assets and Liabilities
- Criminal cases - if any

For our work, we mainly used the personal information of the politician. MyNeta also keeps an internal id (named MyNeta ID) which we keep to resolve duplicates in the data.

Lok Sabha Official data

We also chose to scrape official LokSabha site for politician's personal data as per government records for better triangulation of political entities (especially when we found some information missing in MyNeta dataset) and also for looking up family ties of the political leaders.

We finally got this information from official site of LokSabha which gave us actual government provided records. Scripts were written in scrapy which allowed us to gather information about Indian politicians. We also downloaded the images of politicians which have been used on profile pages of these politicians in the Power Elites web app 3.

Rajya Sabha Official data

We collected Rajya Sabha data as well from their official site to look for more corporate-political tie ups. Rajya Sabha data is more interesting as we have some businessmen directly nominated in Rajya Sabha by our political leaders.

2.2.7 Family Ties Information

We also have manually collected from the web various family ties in businesses and politics. Major source for this data is wikipedia. This data allowed us to find new relations/links from already obtained graph.[5] [13]

All the mash-ups created by these datasets have been reported in viz section 4

2.3 Data Integration

As discussed earlier, the real challenge of forming a data mash-up is the non-uniformity of different sources which include differences in schema, data model, formats of text, assumptions of persons scraping the data(hereby referred to as **data gatherers** or **gatherers**) etc. To alleviate such problems, we came up with a particular data format to be adhered to by all data-gatherers for uniformity before data is pushed to the system.

By separating out the problem of data integration and keeping aggregated data in a separate data-store, we have been able to reduce maintenance costs for the knowledge base. In this way, the schema described in the system model is kept intact, while the data gatherer can crawl the data in accordance with his own whim as long as he conforms to the data format expected by the write API.

The initial plan was to allow the crawled data to be pushed into some MySQL database. Each gatherer would then have to define a mapping somewhere to map all the fields of the crawled data to the fields in the actual graph data. But this poised the problem of maintaining a new map file/table which was crucial for pushing the data. The data gatherer would have to enlist new mappings for every new data crawled from a source. The maintenance of such a database would have been another problem to deal with.
Instead we chose to provide gatherers with a REST API and a specific data format which he/she has to adhere to while pushing in the data. For now, JSON payload is used in a specific format. Very similar to the graph database model, this payload is designed as follows:

```
{
    "taskid": "1",
    "userid": "abhi2@gmail.com",
    "token": "jhfhybfdkjfygfjk87gdfgdjf5",
    "entities": [{
        "id": "1",
        "fetchdate": "1466024344",
        "sourceurl": "http://loksabha.nic.in",
        "labels": ["person", "politician", "entity",
           "businessperson","memberofParliament"],
        "properties": {
```

```
                "name": "Naveen Jindal",
                "address": "Kurushetra",
                "dob": "11 Feb, 1970",
                "job": "business"
            }
        }, {
            "id": "2",
            "fetchdate": "1466024344",
            "sourceurl": "http://loksabha.nic.in",
            "labels": ["\emph{organization}", "politicalparty", "nationalparty", "entity"],
            "properties": {
                "name": "Indian National Congress"
            }
        }],
        "relations": [{
            "id": "1",
            "label": "memberof",
            "start_entity": "1",
            "end_entity": "2",
            "fetchdate": "1466024344",
            "sourceurl": "http://loksabha.nic.in",
            "bidirectional": "False",
            "properties": {
            }
        }]
}
```

The main components of the payload are as follows -

- entities - array with each element having following info
    1. id - unique entity id specific to data gathering task
    2. labels, properties - adhering to an entity in graph db

3. sourceurl, fetchdate - to be used for provenance (to be given by the gatherer) and to be discussed in chapter 3

- relations - array with each element having following info

1. id - unique relation id specific to data gathering task
2. startentity,endentity - ids of participating entities
3. bidirectional - boolean attribute to determine whether the relation is bidirectional
4. labels, properties - adhering to relation attributes in neo4j graph database
5. sourceurl, fetchdate - to be used for provenance (to be given by gatherer) to be discussed in chapter 3

- authentication info - used by the Power Elites app as described in later chapter 3

All data crawled or manually collected is first pushed into a separate graph database (a separate Neo4j instance)- *crawl data-store* (here onwards refer to as crawl-DB) using REST API calls with the JSON payload. The pushed data is then verified and resolved by human moderator before being pushed to the system. A high level view of the central internals of the system is shown in the Figure 2.12.

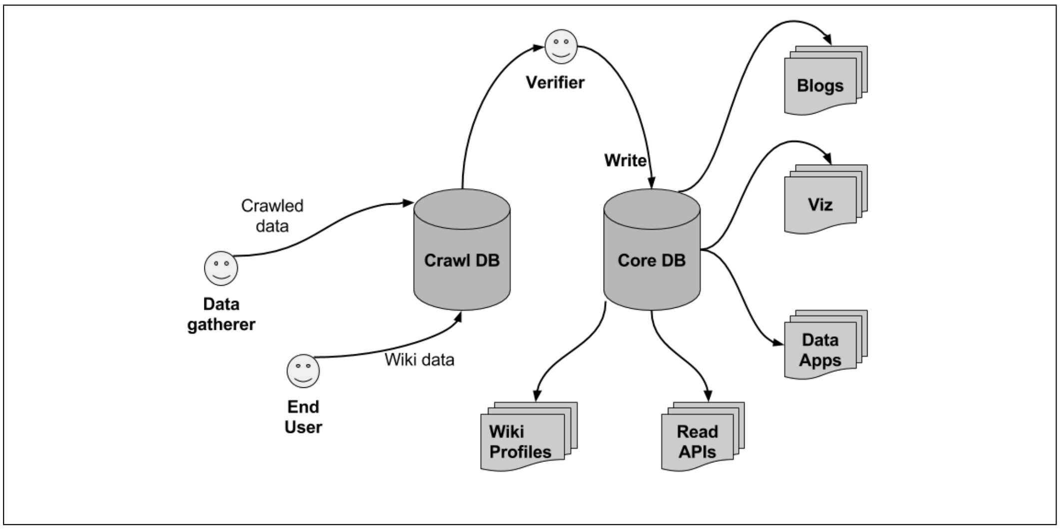


Figure 2.12: A basic model of the system

This way - we have different non-connected subgraphs in the Crawl - DB . But how do we resolve the nodes and relations now with the core data-store? The basic setup is described in the next section of Entity Resolution.

2.4 Entity Resolution

The basic algorithm of entity resolution is as follows-

Algorithm 1: Generic Entity Resolution

Data: Datasets - S, T
Result: list -L consisting of proper matches of T in S
initialization - L is empty ;
parameters - S,T, threshold ;
for *all records t in T* **do**
1 search for t in S ;
2 pick records s in S for which match-score(t, s) $\geq$ threshold ;
3 append all s in L ;
end

Complexity of above algorithm is O(n*m) where n is size(S) and m is size(T) when n records are searched linearly in S. From the above, we can find that the two crucial operations here are the choice of proper scoring algorithm to pick s from S in line 2 and search for t in S in line 1. The faster and better they are, the more efficient is our entire ER task.

2.4.1 String Matching Algorithms

The requirement of a scoring algorithm for our basic ER algo is such that it should match efficiently all similar sounding and similarly spelt words. For our initial testcases, we experimented with Jaro-Winkler distance (and others) as it considers ranges and transpositions while matching two words. A sample experiment and result scores on a list of four names can be seen in table 2.1.

Table 2.1: Comparison of string matching algorithm

Words	levenshtein	jaro_distance	jaro_winkler
vidya vs biday	3	0.78	0.78
saawan kumar vs saavn kumer	3	0.86	0.89
gautam adani vs gautambhai adani	4	0.86	0.92

Upon a test run of all algorithms over a sample of 100 names from our datasets we decided to use jaro-winkler for the purpose.

But, we also needed to match names which are phonetically similar to other names. These helped us to match names like *'Gautam'* and *'Gautambhai'* , *'Vidya'* , *'Bidya'* and *'Biday'* etc.

2.4.2 Comparison techniques

As obvious from the above basic ER algorithm, the performance of entity matching algorithm depends largely on how fast the searching (and thus the comparison between two strings) occurs in dataset S. A naive algorithm like above added a factor of O(n) due to linear search over entire S. As a result, the performance of the entire system got bottlenecked by the resolution module. We started looking for other alternatives regarding this. The obvious improvement over it could have been the possible implementation of a binary search to reduce search speed to O(log n) time. But binary search uses predicates like gt, eq, lt , the value of which are true or false. Such predicates determines some order in the dataset. Data usually cannot be ordered like this eliminating out the possibility of binary search over our data.

2.4.3 Machine learning techniques

After having performance bottleneck in searching records in other datasets, we looked for probabilistic ways of solving the same problem. We used the python library dedupe [6] for this purpose. The library basically induced an active learning mechanism to obtain training samples where it picks two entities of possible match and prompts the user to label positive/negative.

It then matches the related entities accordingly with the hypothesis formed. Unfortunately this approach suffered from following drawbacks-

- Too small data to accurately form a proper hypothesis.
- The labels that were asked to mark along with data were picked at random and often the results of the algorithm are different in different runs (depending on the number of positive or negative label given at that time).

Fortunately indexing the core-dataset proved to a be silver-bullet for all the searching problems.

2.4.4 Indexing and Apache Solr

Indexing allows searching to be very fast - of near O(1) speed. Indexes are data-structures that store contents of a document (in our context fields of records in a dataset) for faster access to the document. This enables faster retrieval with a trade-off of using more memory. For our purpose we used Apache Solr framework [2] for the searching step. Solr uses Apache Lucene [1] to create an inverted index on desired fields in the records. An inverted index basically creates a data structure on the content of the records and have pointers to the actual locations of the records. So for all text in the specified fields, Lucene breaks them (tokenizes them) and store them in a data structure for fast retrieval. Solr also sets up a Web server with REST APIs to allow us to integrate it with other parts of our system described in Chapter 3. Since Solr has its own sets of protocols we had to modify the way we apply the resolution algorithm described above. The main steps followed by us to realize this are as follows.[3]

- **Defining a data set** to index (the data set S in ER algo). We described a data source which contained the dataset. In this case, it was a mysql database. (file *db-data-config.xml*) .
- **Defining the fields** to index. Solr needs to have a schema of the records to know which fields will indexed and which one is kept as

satellite data. These are specified in solr configuration files. (file *schema.xml*)

- **Defining how to pre-process** all the terms before creating the index. Solr allows to specify a list of pre-built tokenizers, stemmers, filters to preprocess a term or custom ones if necessary. We used whitespace tokenizers and double metaphone phonetic filters to get a match score relevant to our purpose and in this way used the phonetic algorithm for better text matching. (file - *solrconfig.xml*)

- **Forming a lucene query** based on the contents of records of another dataset (data set T above)

- **Applying Jaro algorithm** for comparison was difficult in Solr. This is because, all the functions applied to the text for indexing are necessarily single parameter. Jaro or any other non-phonetic string metric needs at-least two strings. Results returned by the Solr can be further filtered by the Jaro-Winkler algorithm. Results being quite small, does not take much time even if a one-one matching algorithm is executed.

Solr Fields

To effectively triangulate two entities, special emphasis was given on which fields to compare while doing it. After few experiments we decided to match records on following fields generated from the graph data model we discussed in previous section.

- **Aliases**- a list that contains alternate and primary names pertaining to an entity. This is especially required when a person/institution is known by several names in the world. Eg. - Narendra Modi vs Narendra Damodar Modi, BJP vs Bharatiya Janata Party

- **Aliases Phonetic** - same as above but here search is done on phonetic index.

- **Label** - Label dictates the type of entity as per data model in section-2.1. An entity can have multiple labels.
- **Keywords** - a list that contains main keywords of the entity. This field is most helpful in triangulation as it indexes all the properties of the entity and the aliases of the entities directly related with the original entity. Important properties unique to a particular entity like location for a particular item.

Lucene Queries

Proper search queries are essential for efficiently resolving an entity. These involves using the above indexed fields effectively to obtain relevant entities. The grammar of the lucene query used goes as follows-

$\boldsymbol{L}$ ::= $\boldsymbol{I}$ ':'($\boldsymbol{Q}$) $\boldsymbol{L}$ *OR* ϵ
where, $\boldsymbol{I}$ ::= *<index field >*
$\boldsymbol{Q}$ ::= *<query string>to be matched against I*

query string can be as follows -

"string text" - matching the exact word 'string text'
string text - matches string or text or both
string~ - lucene applies edit distance to string and returns the possible matches

More details on lucene query can be found at `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`

Performance

On testing our Solr integrated system against the original basic resolution algorithm, we found many-folds improvement. Initially upon resolving 1000 corporate entities against about 500 political ones over the "name" field in both the datasets took us about 150 minutes to resolve using basic ER algo. That makes resolution of a single entity about 18s. Compared to that, a single entity is resolved in Solr in little over 1s showing a performance upgrade of almost around 18x.

Chapter 3

Design of Power Elites Web App

In the present chapter, we dissect our system bit by bit and describe each component in detail. We have already described how our data is integrated from different data sources, and also how our ER system works. Here we describe the wiki and the web application that uses all those features. It is important to note that that our main goal while designing the web application (and the entire system) was to reduce manual work of the verifier - so a lot of brainstorming and design changes went in that direction. Any feature or performance improvement in the verification process makes life easy for the verifier. The system has been designed to be able to crowd-source the verification process.

Another key point which we have kept in mind while building the system is to make the data-gatherers' task hassle-free - that they should universally be able to communicate with the system. Towards this effect they should also be able to keep track of their data, and update their old data as well. We have already described how we achieve this using JSON and REST over a Neo4j back-end which we call crawl data-store. We have already described how separation of concerns further help in achieving this as the crawled-data is well separated in crawl data store.

3.1 Terminology

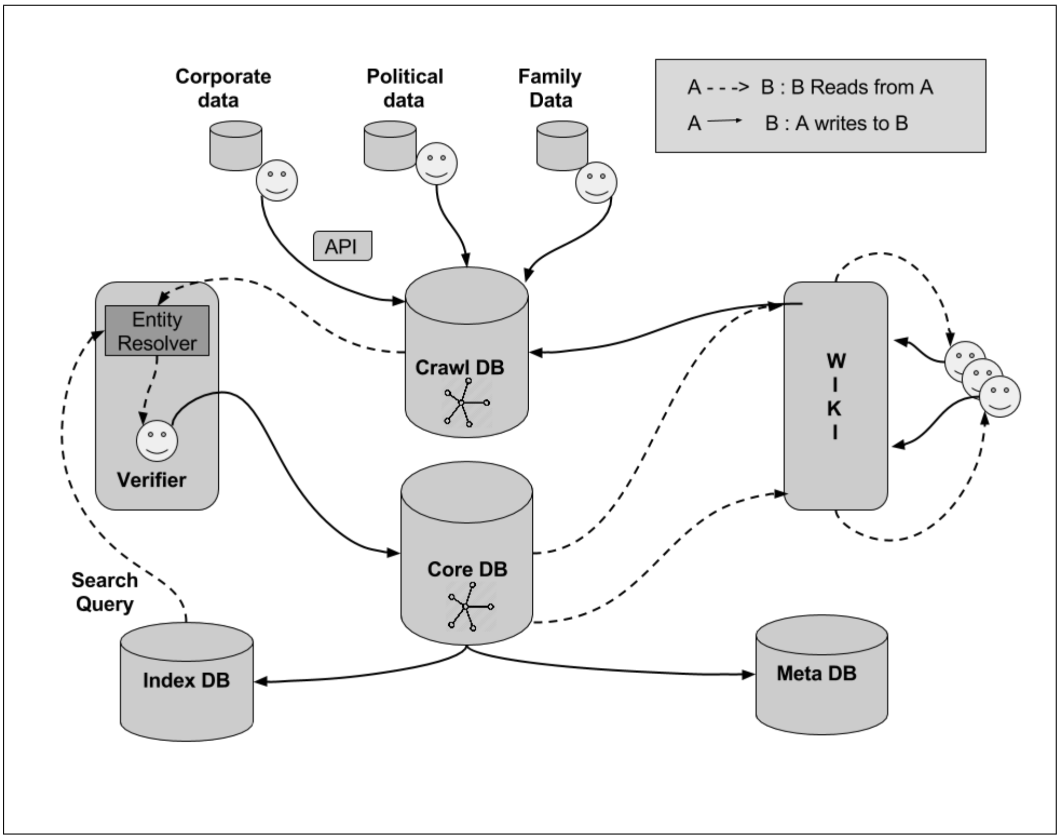


Figure 3.1: System Design

The detailed design of the system can be seen in 3.1. We formally define the terminology about the actors and the system components here:

1. **Data-Gatherer:** An authorized individual or a group of individuals who crawl linked data from different sources. This data can be about entities and relations that belong to a domain supported by our system: political, corporate, sports, bureaucratic, etc. A data-gatherer can use their API-token to push data to our crawl data store using JSON payload.

2. **Crawl Data Store:** The Neo4j data store which stores all non-verified, non-resolved data that data-gatherers push into the system. All data from here is first verified and resolved.

3. **Resolver:** A tool that searches over indexed graph data to suggest possible matching entities for a given entity. ER mechanism has been described in Section 2.4. A resolver in essence in our system suggests the probable matches but does not resolve.

4. **Verifier:** An authorized person who matches an entity against a possible list of suggestions by the resolver. The only human element on which the system is dependent to be able to push correct data to the core data store.

5. **Core Data Store:** The Neo4j data store which stores all verified, resolved data that data-gatherers push into the system. This basically represents our knowledge base - an integration of data from different sources.

6. **Registered User:** An authorized user who can use wiki to suggest changes to an entity or a relation in the core data store. All these changes are directed to crawl data store.

7. **Wiki:** Part of the web app, where registered users can edit or add information to the core data store.

8. **Meta-DB:** MySQL back-end which stores provenance of any info added or changed in the core data store.

9. **Index-DB:** MySQL back-end which stores condensed information (and connected information) of all the entities in the core data store. Apache Solr runs on top of it to index this information to speed-up the resolution process.

10. **Admin:** The user with all the privileges in the system - can delete all indexes, refresh all indexes, see which user/verifier contributed most to the system, change role of a user, etc.

11. **End user:** Non-authorized users that can access information through the web app. They cannot make or suggest any changes to the core data store.

12. **Role:** All the authorized users in the system are given a role. The roles are in a scoping fashion. The role order from the top is as follows: Admin, Verifier, Crawler, Registered User, End User.

3.2 Data Gatherer

The use case diagram for data-gatherer is shown in Figure 3.2.

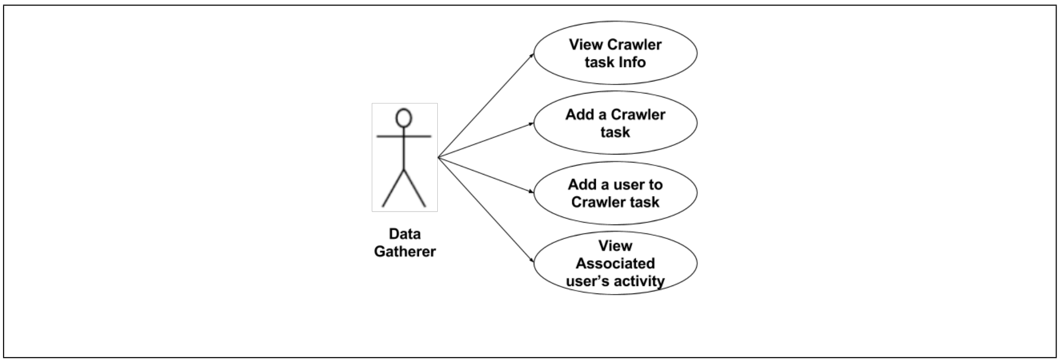


Figure 3.2: Use cases for Data Gatherer

3.2.1 Task creation

A group of data-gatherers are assigned a unique task identifier when they create a task for their crawled data (Figure 3.3). This helps in separating the data from different tasks, due to which the crawl data-store has different disconnected subgraphs. The data-gatherers can keep track of their pushed info using the task identifiers (Figure 3.4). Each node and each relation for a task has to be uniquely named by the data-gatherers relative only to their task id.

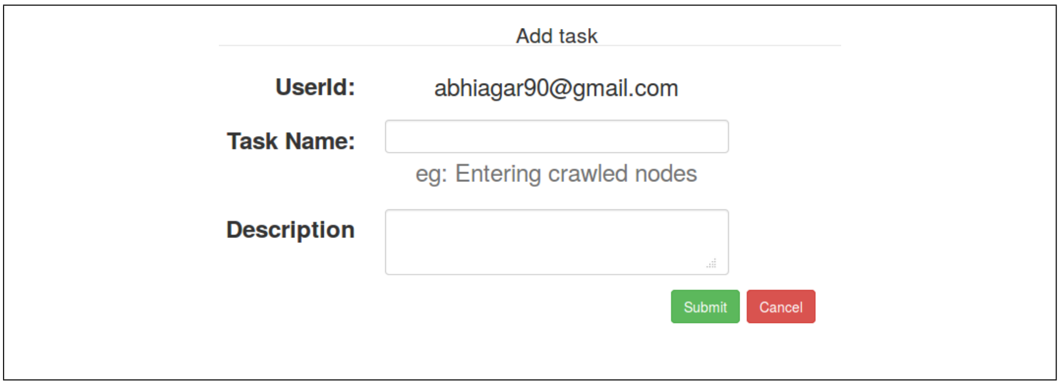


Figure 3.3: Creating tasks for a data-gathering group

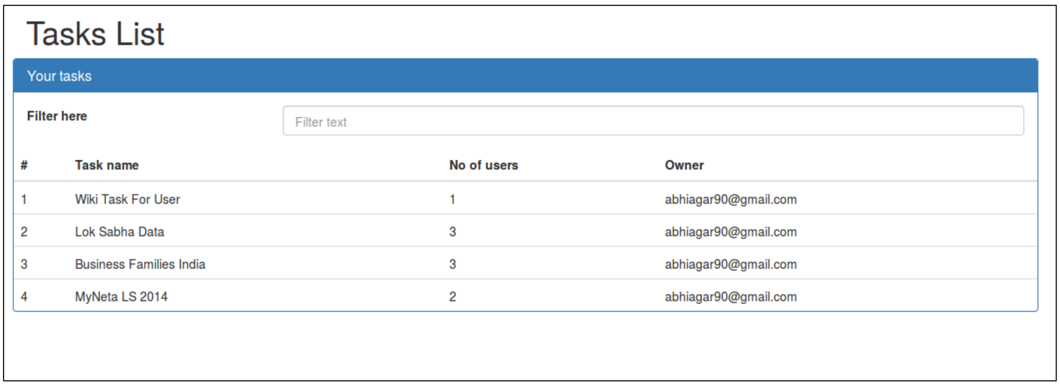


Figure 3.4: Information about a task

3.2.2 User Addition

By default when a task is created, the owner is automatically added as one the users of the created task. He can add more users to the task which will allow those users to be able to push data specific to that task (Figure 3.5).

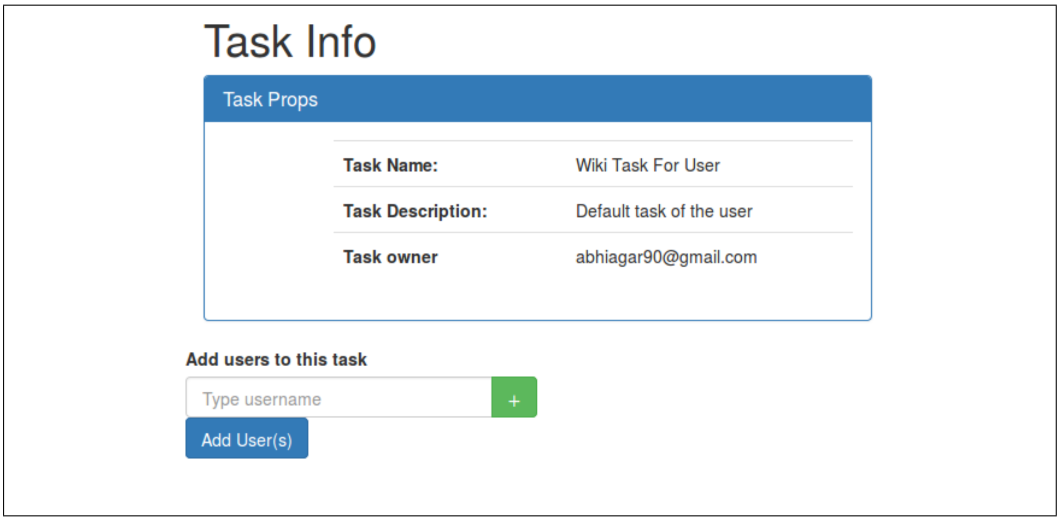


Figure 3.5: Adding users to a task

3.2.3 Authorization & POST API

Every authorized user in the system is given an api-token at the time of sign-up. An api request is authorized if userid and api-token are provided correctly. For pushing data to the system using a valid api-request, JSON has been employed as the payload. The structure has been described in Section 2.3. To push the JSON to the system, the API request is described in Table 3.1.

Table 3.1: Request API: /apis/postgraph/?userid=$\langle$*userid*$\rangle$&token=$\langle$*tokenid*$\rangle$

URL Parameters	userid, token
Headers	Content-Type : application/json
Payload	JSON
JSON VARS	taskid, userid, token, entities, relations
HTTP-METHOD	POST

3.2.4 JSON Response

If request is authorized and JSON is as per our convention, then the data is pushed to the crawl data-store. Validations before the push happen as described in the 2.3. The response is almost identical copy of the request JSON, including the meta-data. All the meta-data has been marked with a beginning and an ending underscore as shown in 3.2.4


```
"16": {
      "labels": [
        "person",
        "entity",
        "indian",
        "politician"
      ],
      "properties": {
        "_crawl_en_id_": "en_7_16",
        "_fetchdate_": 1466781745,
        "_nodenumber_": 16,
        "_pushdate_": 1467056465.732715,
        "_pushedby_": "mridul.goel53@gmail.com",
        "_sourceurl_": "https://www.wikipedia.org/",
        "_taskid_": 7,
        "name": "Akhilesh Yadav"
      }
    }
```


3.3 Resolver and verifier

1. **Use Case:** The use case diagram for verifier is shown in Figure 3.6

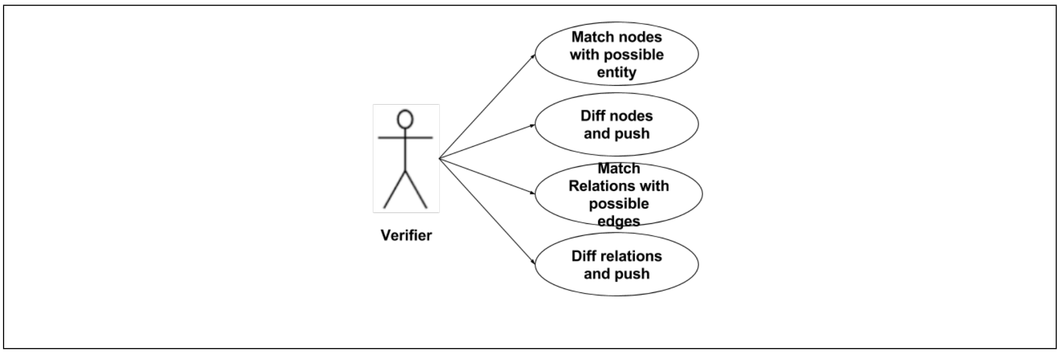


Figure 3.6: Use cases for Verifier

2. **Resolution:** Every node in the crawl data-store has to be resolved to a node (existing or new) in the core data-store. To resolve an entity is to provide it with a uuid. Similarly for relations, a relid is provided. This is achieved when a verifier picks up an object during match (Figure 3.7) from the possible list of objects suggested by the resolver. Diff view (Figure 3.8) aids the verifier to granularly look at new labels, new properties and conflicting properties (shown with differentiating colors).

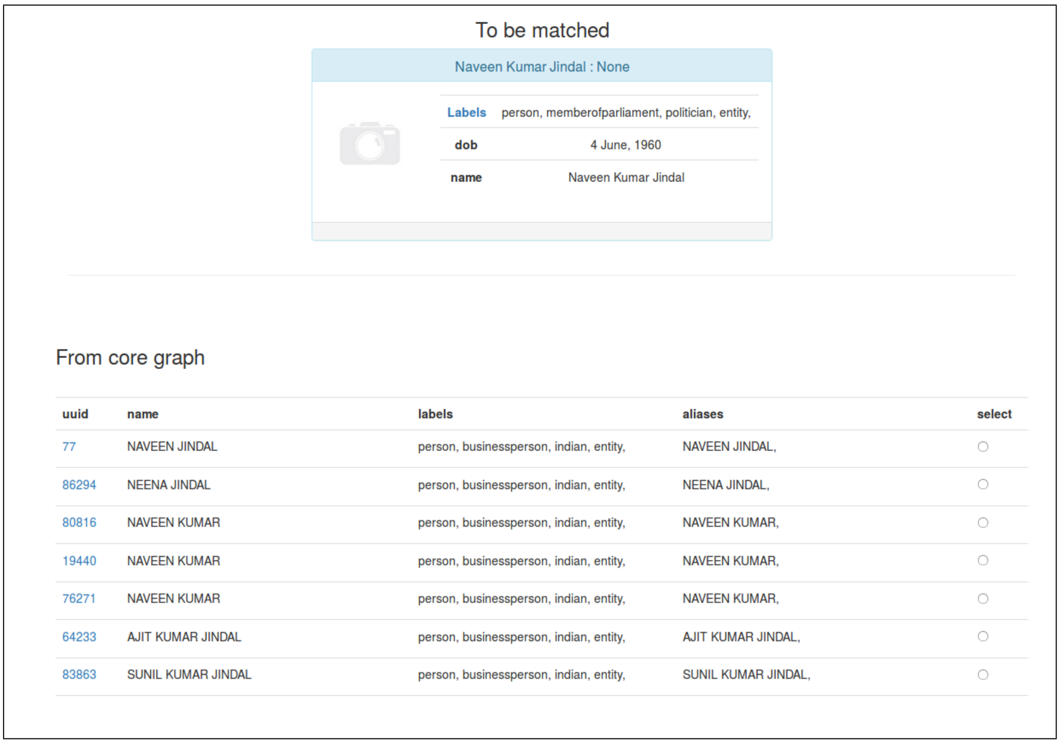


Figure 3.7: The match-view in verification task

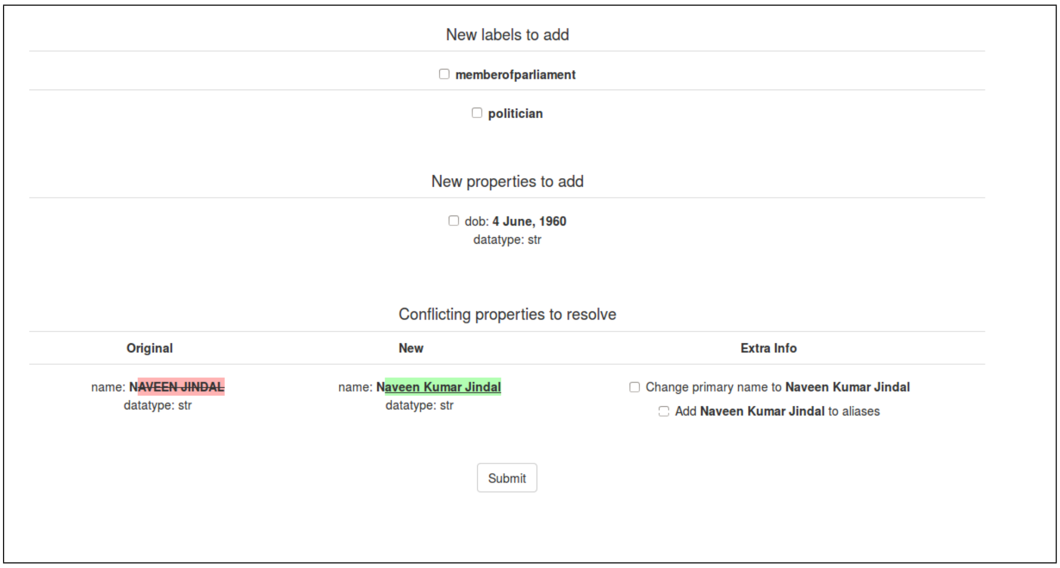


Figure 3.8: The diff-view in verification task

3. The key idea is to design views to facilitate the verification task for the verifiers, to help them in focusing on investigating the new information in the least possible time.

4. **Selection algorithm:** A selection algorithm picks up the next nodes or relations to resolve. When left with no choice, it picks up the highest degree node, else the node that has the highest number of connected resolved nodes is picked.

5. **Validations**: Robust validations have been used on match and diff view to ensure that nothing inconsistent happens to the core data-store. Option to resume an on-going task, clear current session, release all locks have also been provided (Figure 3.9).

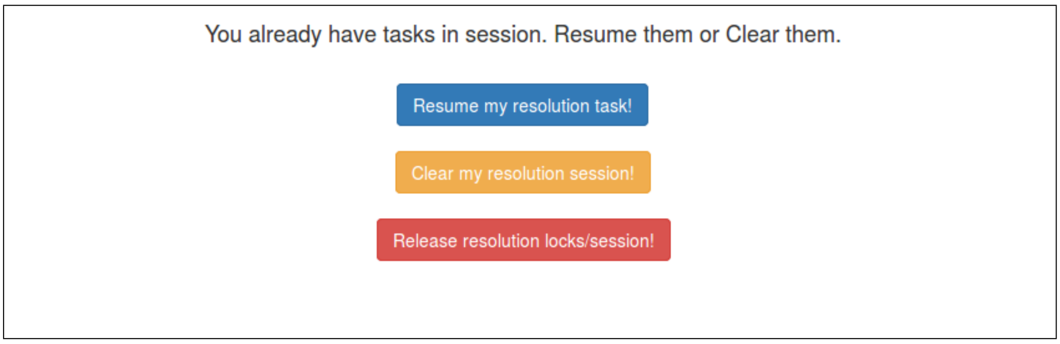


Figure 3.9: Session Handling during Verification task

6. **Parallelization:** A graph object selected during a verifier's session is locked for some time to let the selection algorithm know of it's status, this way the selection algorithm picks up another potential node for a new user.

7. **Jaro:** Provision for running jaro on fetched entities from Apache Solr has been provided in the match view. In practice, phonetic has been seen to be do very well for names we have encountered. We have kept this feature extensible in the sense if some other algorithms need to be tested for the task.

8. **Multi valued-properties:** Extensive care has been taken to incorporate multi-valued property for nodes and relationships. During the diff task, a merge request on the property can result in converting the original value of the property to a multi-valued one.

3.4 Authorized User and Wiki

1. **Use Case:** Use case diagram for authorized users is shown in Figure 3.10.

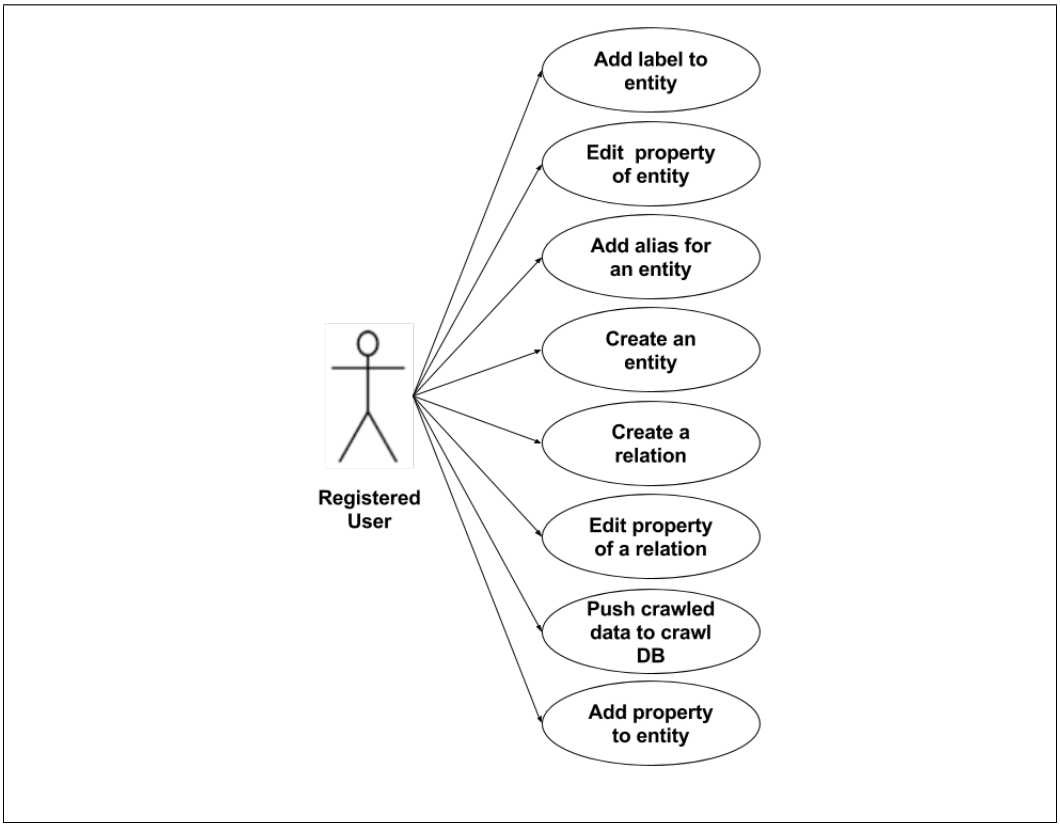


Figure 3.10: Use cases for registered users.

2. **Wiki:** Authorized users can access the wiki to edit entities and relationships in the core data-store. All these edits are first sent to the crawl data-store in the same fashion as any JSON from a data-gatherer is handled. This way all changes in the system occur through the verification process so that the core data-store is free of any noise or redundancy.

3. **Wiki features:** Wiki can be accessed to add a new node, add a new relation (Figure 3.11), edit an existing node, add a new label (Figure 3.12), edit an existing relation. A user can add as many properties to the entity/relationship as possible (Figure 3.13). Certain properties that can be potential candidates for addition are also suggested in the edit view (Figure 3.13).

Figure 3.11: Add relation between two entities

Figure 3.12: Add new labels to a node

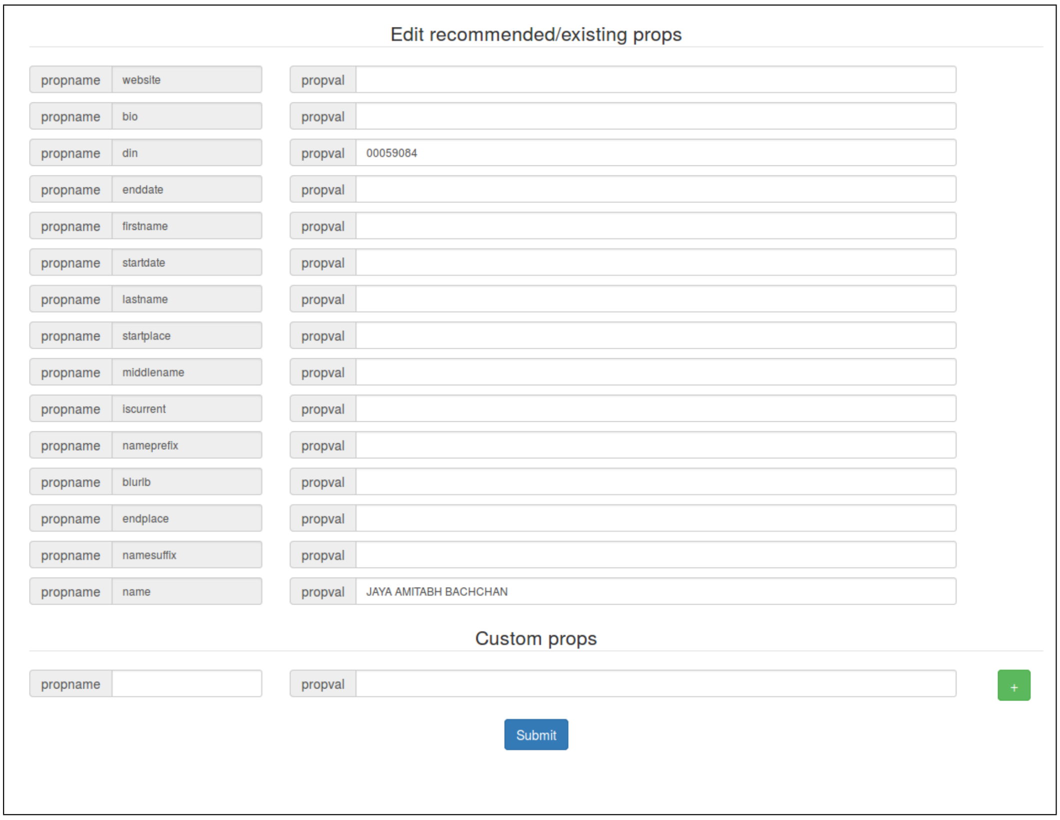


Figure 3.13: Edit existing properties/Add recommended properties

3.5 Provenance

1. **Accountability:** We felt the need to maintain every granular detail about any change in the core data store. Thus, a label addition, a property change, a property addition, all need to be accounted for - who pushed the change, when was it pushed, when was the data collected by the data-gatherer, who verified the change, the source-url for the new information, etc.

2. **Meta-DB:** All this meta-data is stored on a dedicated MySQL back-end that is updated as soon as a new updation is approved by the verifier.

3. **History:** A history feature on the web application for each entity and for each relation helps an end user to see our provenance meta-data. The same is better viewed in Figure 3.14 and Figure 3.15.

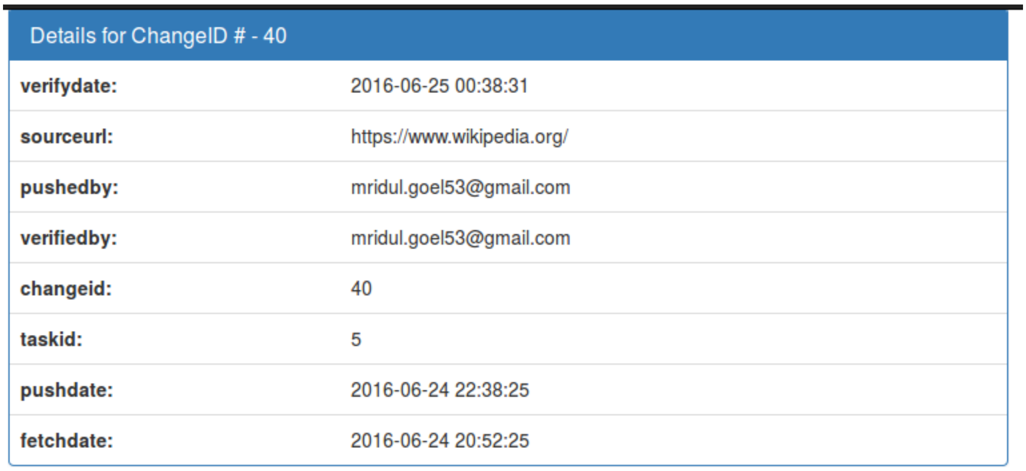
Labels for MUKESH DHIRUBHAI AMBANI

uuid	changetype	changeid	label
124	1	1	person
124	1	1	indian
124	1	1	entity
124	1	1	businessperson

Properties for MUKESH DHIRUBHAI AMBANI

uuid	changetype	oldpropvalue	newpropvalue	propname	changeid
124	2	[u'MUKESH DHIRUBHAI AMBANI']	[u'Mukesh Ambani', u'MUKESH DHIRUBHAI AMBANI']	aliases	40
124	2	MUKESH DHIRUBHAI AMBANI	Mukesh Ambani	name	40
124	1		00001695	din	1
124	1		124	uuid	1
124	1		MUKESH DHIRUBHAI AMBANI	name	1
124	1		['MUKESH DHIRUBHAI AMBANI']	aliases	1

Figure 3.14: Every change is attributed to a change ID



Details for ChangeID # - 40	
verifydate:	2016-06-25 00:38:31
sourceurl:	https://www.wikipedia.org/
pushedby:	mridul.goel53@gmail.com
verifiedby:	mridul.goel53@gmail.com
changeid:	40
taskid:	5
pushdate:	2016-06-24 22:38:25
fetchdate:	2016-06-24 20:52:25

Figure 3.15: Every change has associated meta-data

3.6 End user

1. **Use Case:** Use case diagram for not logged-in end users is shown in Figure 3.16.

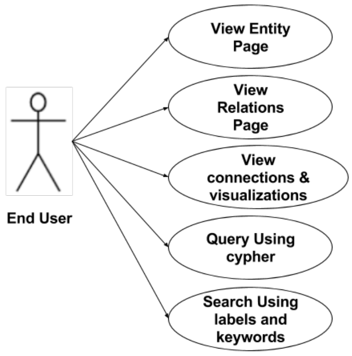
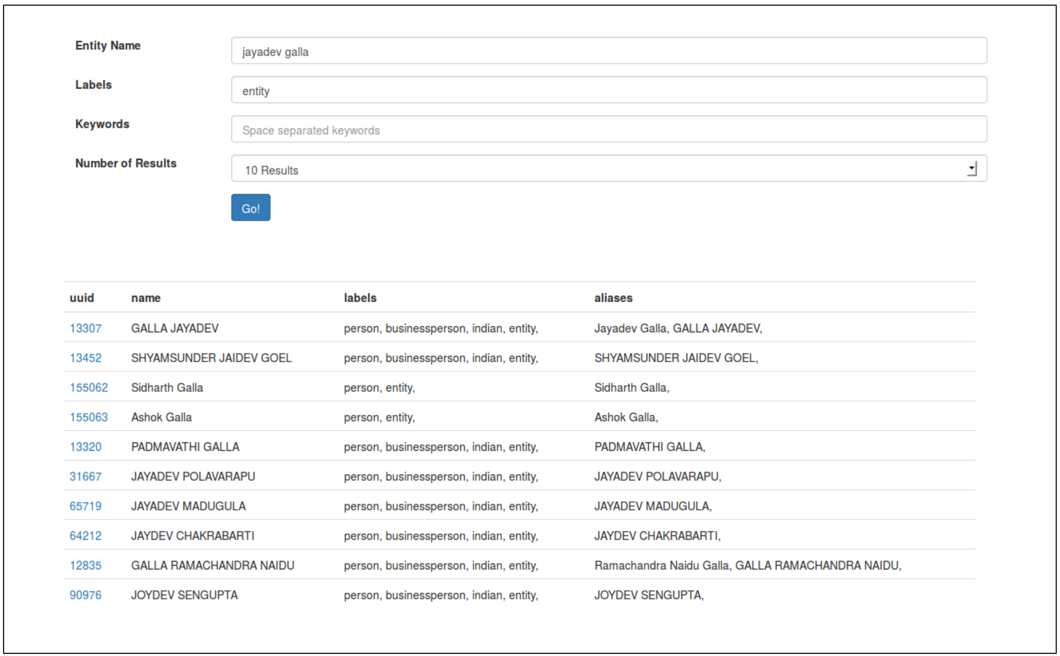


Figure 3.16: Use cases for End User

2. **Search:** End user can search for entities in our core data store. The search is powered by Apache Solr and uses double metaphone. The search can be filtered on labels and keywords. An actual search query is shown in Figure 3.17



uuid	name	labels	aliases
13307	GALLA JAYADEV	person, businessperson, indian, entity,	Jayadev Galla, GALLA JAYADEV,
13452	SHYAMSUNDER JAIDEV GOEL	person, businessperson, indian, entity,	SHYAMSUNDER JAIDEV GOEL,
155062	Sidharth Galla	person, entity,	Sidharth Galla,
155063	Ashok Galla	person, entity,	Ashok Galla,
13320	PADMAVATHI GALLA	person, businessperson, indian, entity,	PADMAVATHI GALLA,
31667	JAYADEV POLAVARAPU	person, businessperson, indian, entity,	JAYADEV POLAVARAPU,
65719	JAYADEV MADUGULA	person, businessperson, indian, entity,	JAYADEV MADUGULA,
64212	JAYDEV CHAKRABARTI	person, businessperson, indian, entity,	JAYDEV CHAKRABARTI,
12835	GALLA RAMACHANDRA NAIDU	person, businessperson, indian, entity,	Ramachandra Naidu Galla, GALLA RAMACHANDRA NAIDU,
90976	JOYDEV SENGUPTA	person, businessperson, indian, entity,	JOYDEV SENGUPTA,

Figure 3.17: Search Page results

3. **View entity profiles and connections:** User can browse to profile pages of entities in our core data-store. The profile shows information about the particular entity and also displays first level relations for the same. An actual profile from the core data-store and connections for the same entity are shown in Figure 3.18 and 3.19 respectively.

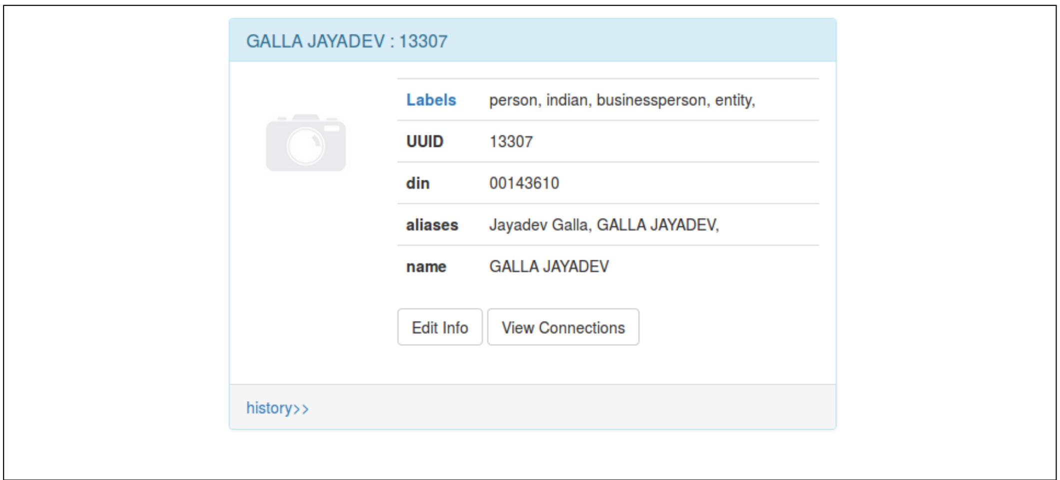


Figure 3.18: Entity profile from core data-store

All connections

relation	relationship with [entity]	relid
- worksin - >	AMARA RAJA INDUSTRIAL SERVICES PRIVATE LIMITED - 152989 [organization, company, indian, entity,]	63118
- worksin - >	AMARON BATTERIES PRIVATE LIMITED - 147962 [organization, company, indian, entity,]	63119
- worksin - >	RNGALLA FAMILY HOLDINGS PRIVATE LIMITED - 129709 [organization, company, indian, entity,]	63120
- worksin - >	NINE NINES LIFESTYLE PRIVATE LIMITED - 117342 [organization, company, indian, entity,]	63121
- worksin - >	AMARA RAJA INFRA PRIVATE LIMITED - 110678 [organization, company, indian, entity,]	63122
- worksin - >	AMARA RAJA POWER SYSTEMS LIMITED - 107945 [organization, company, indian, entity,]	63123
- worksin - >	AMARA RAJA ELECTRONICS LIMITED - 108204 [organization, company, indian, entity,]	63124
- worksin - >	MANGAL INDUSTRIES LIMITED - 99029 [organization, company, indian, entity,]	63125
- worksin - >	GALLA FOODS LIMITED - 99149 [organization, company, indian, entity,]	63126
- worksin - >	AMARA RAJA BATTERIES LIMITED - 94475 [organization, company, indian, entity,]	63127
- relatedto - >	GALLA RAMACHANDRA NAIDU - 12835 [person, businessperson, indian, entity,]	231267
- relatedto - >	Aruna Kumari Galla - 155061 [person, entity,]	231269
< - relatedto -	Ashok Galla - 155063 [person, entity,]	231270
< - relatedto -	PADMAVATHI GALLA - 13320 [person, businessperson, indian, entity,]	231271
< - relatedto -	Sidharth Galla - 155062 [person, entity,]	231272

Figure 3.19: Entity connections from core data-store

4. **Visualizations for the end user:** The end user can generate visualizations for a cypher query. The cypher query is first validated for safety and load, and then executed. Result of an actual cypher query is shown in Figure 3.20.

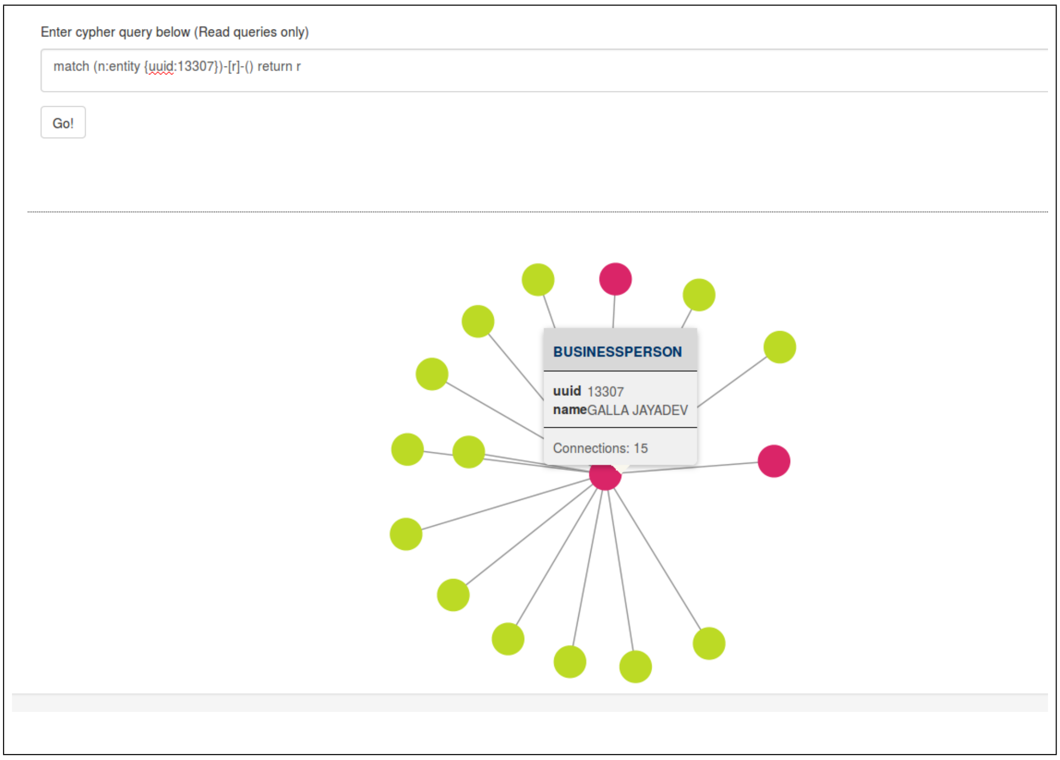


Figure 3.20: End user generates visualization

3.7 Admin

1. **Use Case:** Use case diagram for not logged in end users is shown in Figure 3.21.

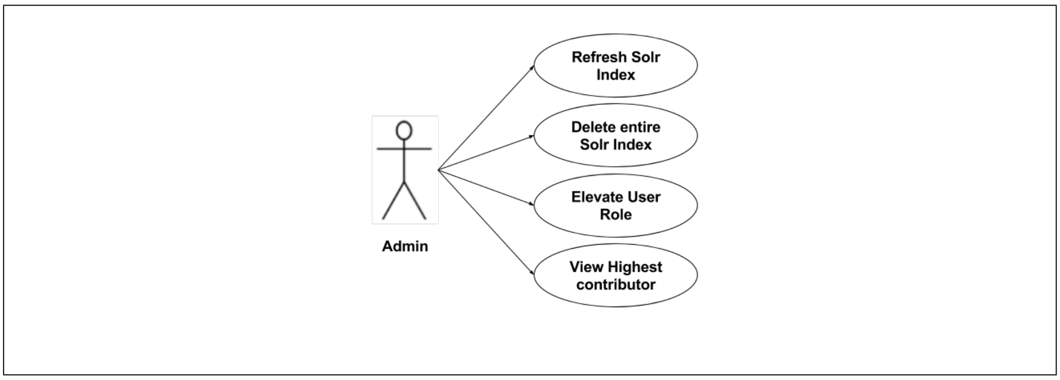


Figure 3.21: Use cases for Admin

2. **Solr Indexes:** Admin can delete all the Solr indexes, and can also

refresh them for maintenance purposes.

3. **Change user roles:** Admin can change roles of existing authorized users, so a user roles in the system are always controlled by the admins.

4. **View contributions:** Admin on a broader level needs to look over the verification/moderation process in the system. Towards that effect, an admin can view all the statistics about verifiers and data-gatherers present in the system.

5. **Bots:** Not all the tasks can be done by humans alone, and thus we have introduced the concept of bots in the system. All these bots wake up at regular intervals and execute any background jobs scheduled for them. All these bots thus need to have admin roles for this purpose. We have a *Location resolver bot* that periodically checks if any entity has an address field and is not connected to any city. It then automatically parses the address and introduces a relation from that entity to that city. We also have a *Dangling locks release* bot that releases any dangling locks in the crawl data store after a specified interval.

Chapter 4

Visualizations and Analysis

When we started out, the initial core data set contained mostly director sharing data. The process of data collection for this dataset is described in data collection section 2.2. Interesting insights come about when we dig inside this dataset and try to find out how a big entity is connected to another big entity.

Interesting results also come about when we merged this dataset with family trees and political data that we obtained. Though this list is a bit long, we have tried to document only the best examples here.

We have just been able to touch three realms here: corporate, political, entertainment. Also, we have tried to see family trees to validate nepotist practices in politics and business. The idea through these visualizations is to show that the power houses interact and mingle mostly among themselves.

4.1 Influence Network

Here we describe the case of politician and business tycoon Naveen Jindal. He is directly on the board of **8** companies. We define here a hop as company-director-company link. Through director sharing, in **2 hops**, he spans **144** companies, in **3 hops** the span network is greater than **1800** companies. Interestingly in **4 hops**, his influence network reaches more than **13000 companies out of the 60000 companies** network we have in our DB! Naveen Jindal is connected to (within 3 hops): *Ashok Leyland, Ambuja cements, Reliance Power, Indiabull, ONGC, JP Associates, Idea, Essar, IDFC, Tata Motors, Shriram, Bharti, Max Life, IDBI, Mahindra, BPCL, Network 18, BHEL, Lanco, Maruti Suzuki, ICICI Bank, Infosys, Adani, ITC, Tech*

Mahindra, Wipro, HDFC, Hinduja, DLF, Indraprastha Gas, Dabur India, Jet Aiarways, JK Lakshmi Cement ad nauseum. It seems as if every big company is connected to the other. Adding to that, he was an MP for ten years.

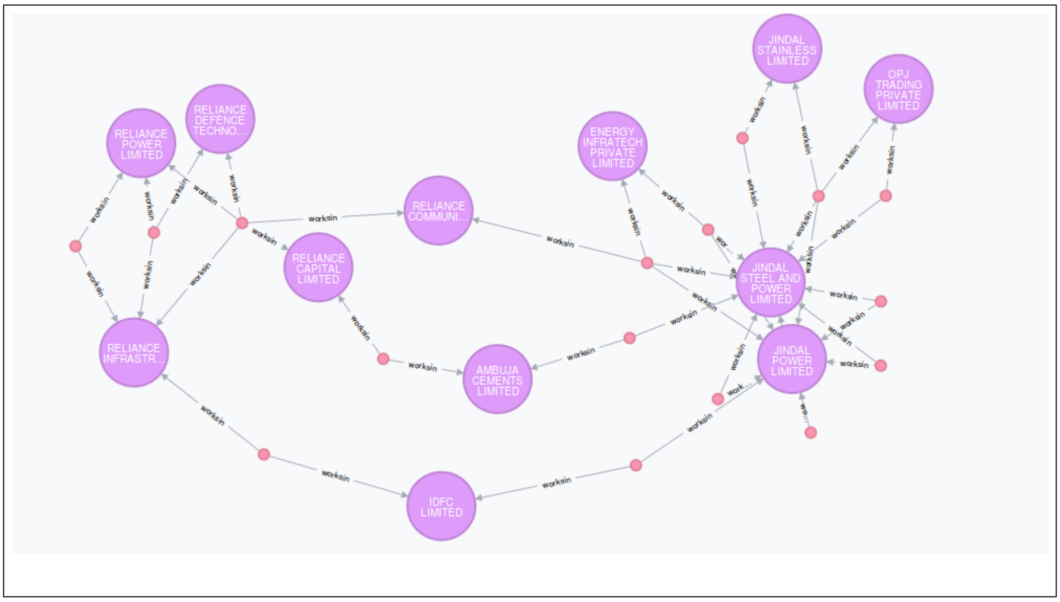


Figure 4.1: Naveen Jindal's connections with Reliance

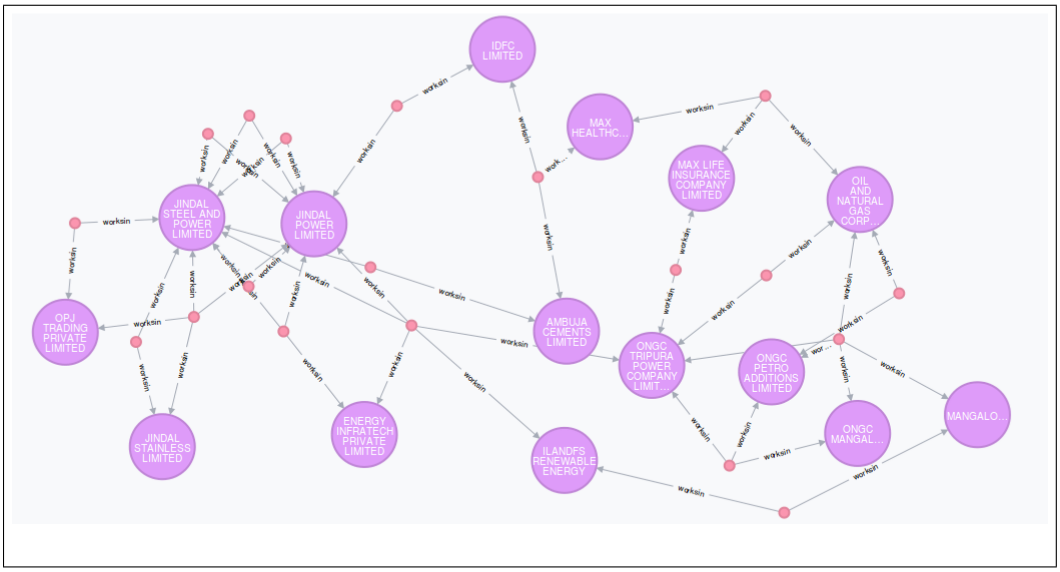


Figure 4.2: Naveen Jindal's connections with Ambuja Cement and ONGC

4.2 Interesting directors

4.2.1 Not just Mahindra

Figure 4.3: Deepak Parekh and his first-level Companies

4.2.2 The Lavasa Connection

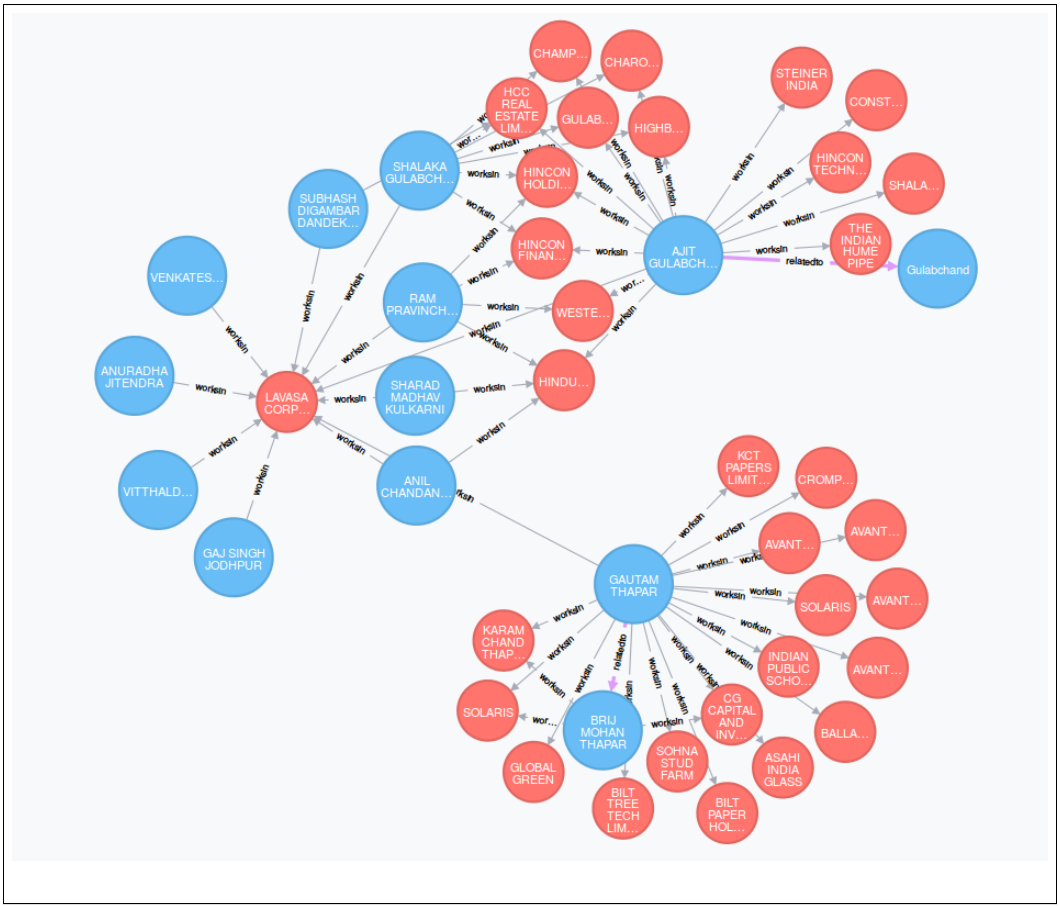


Figure 4.4: Gulabchands and Thapars

4.3 Media Houses

4.3.1 Birlas, HT and Ambanis

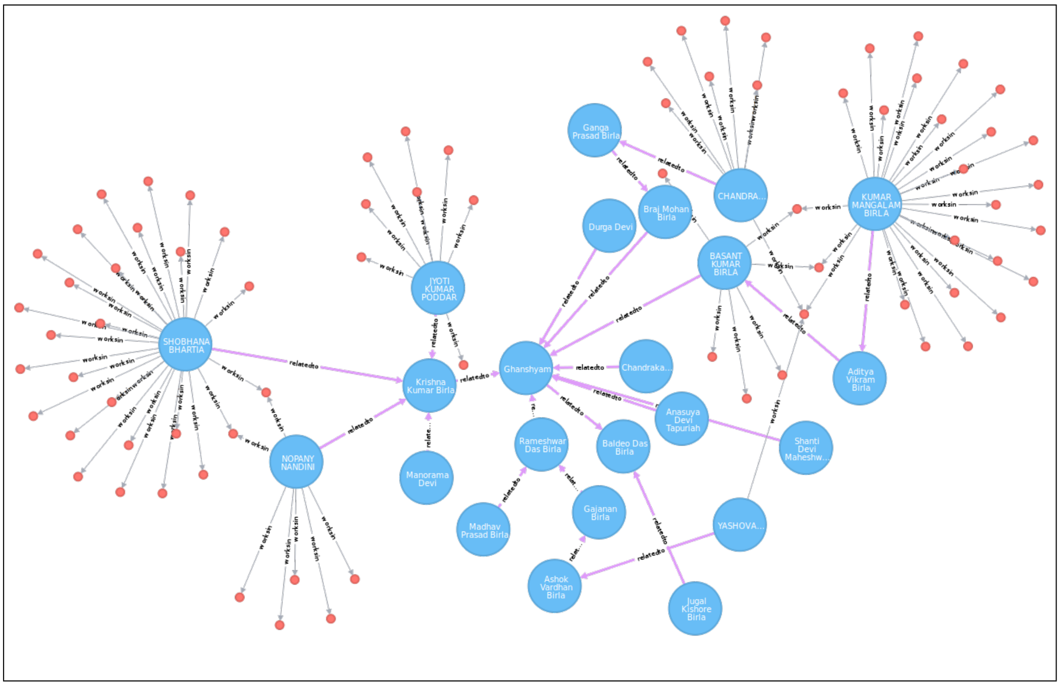


Figure 4.5: Shobhana Bhartia - Hindustan Times owner, ex-Rajya Sabha MP

4.3.2 The Sarkars

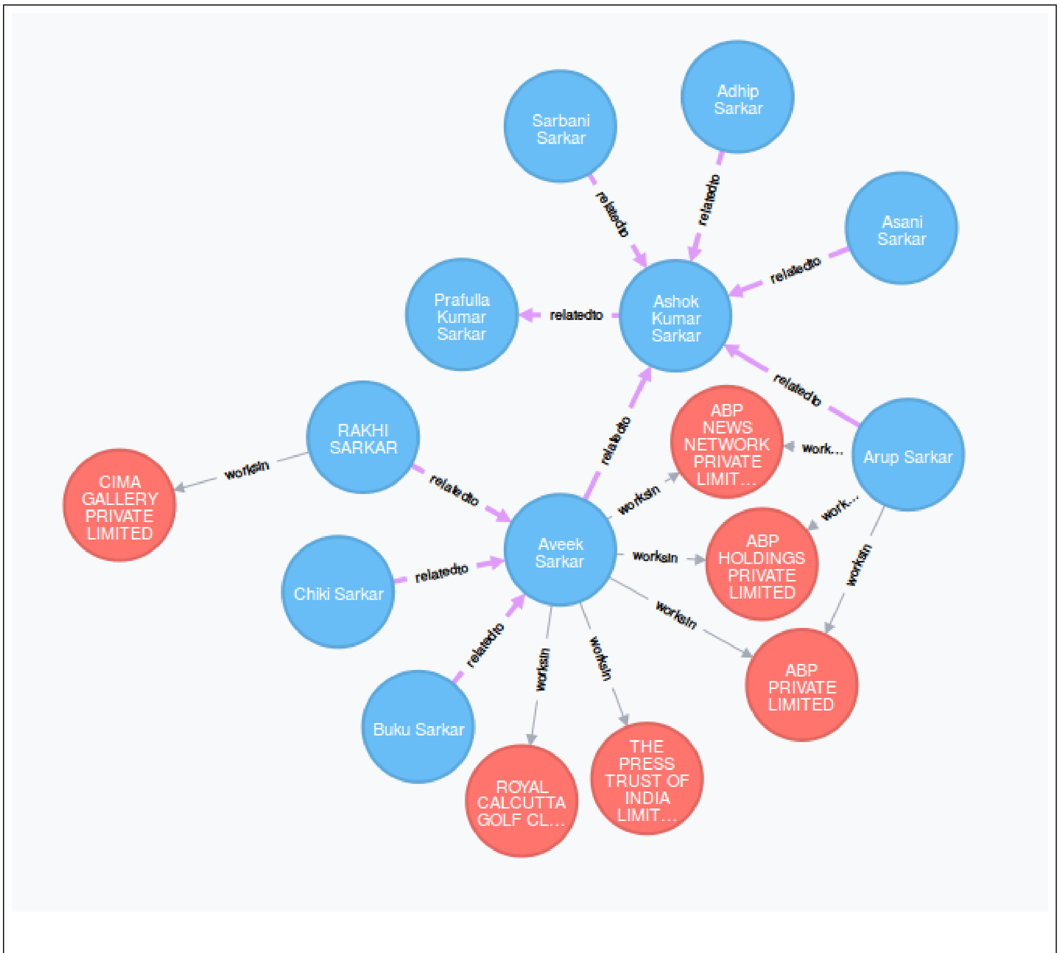


Figure 4.6: Aveek Sarkar of the Telegraph and ABP group

4.4 Family trees

4.4.1 Ambani's

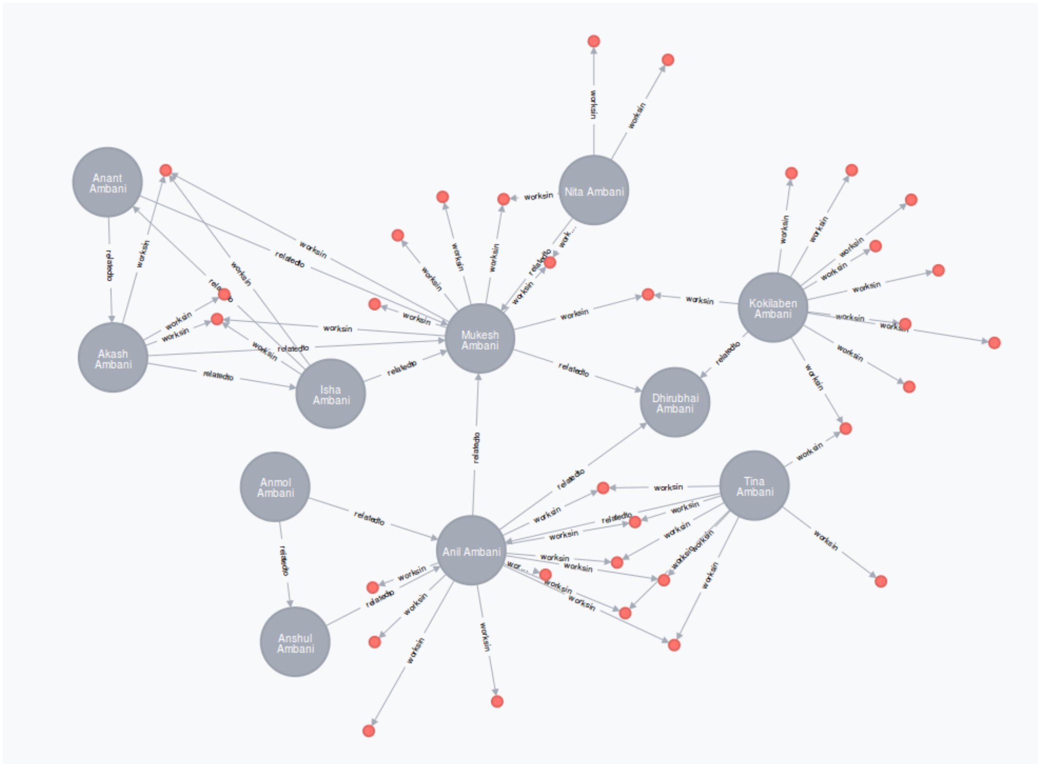


Figure 4.7: Reliance: It's all in the family

4.4.2 The better halves

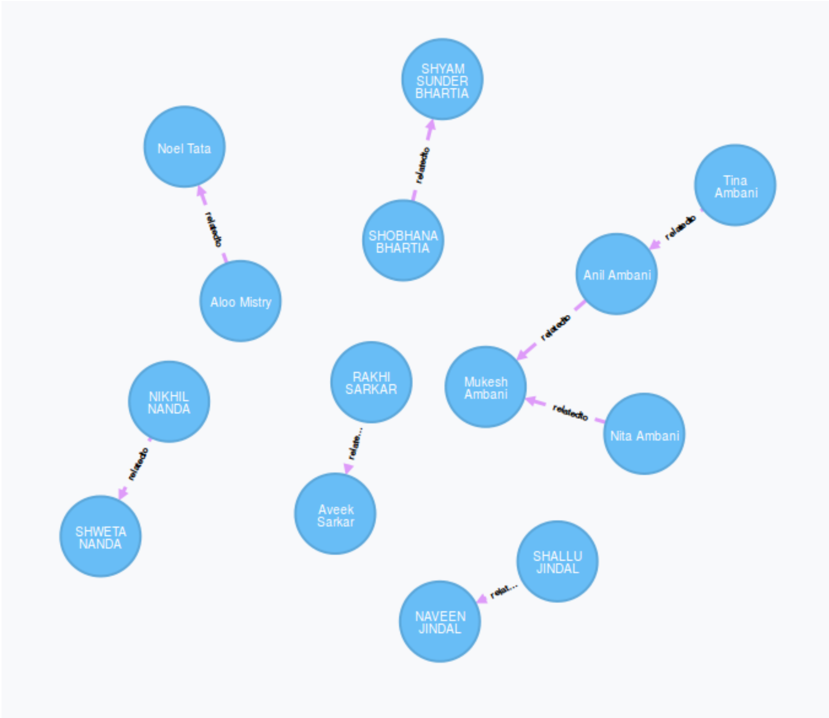


Figure 4.8: Businesswomen and also spouses

4.4.3 Yadavs from UP and Bihar

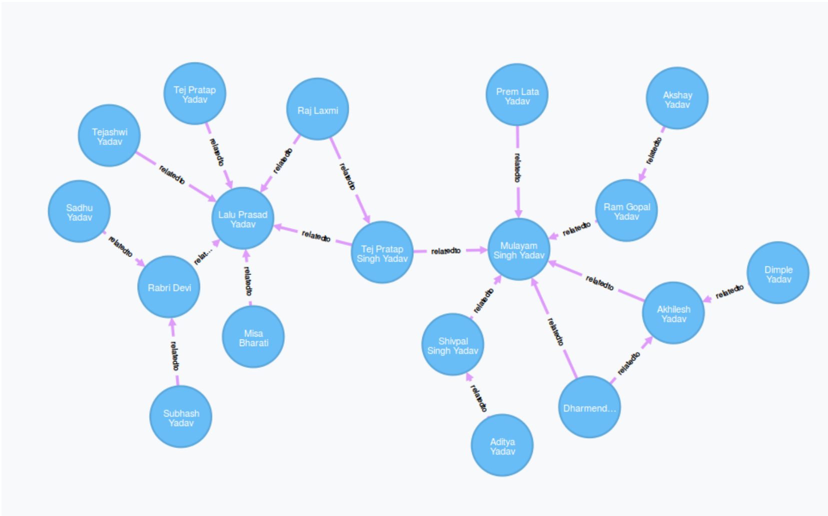


Figure 4.9: Connection between Lalu Prasad Yadav and Mulayam Singh Yadav

4.5 Runaways or corporate hulks

4.5.1 Modis of Modinagar

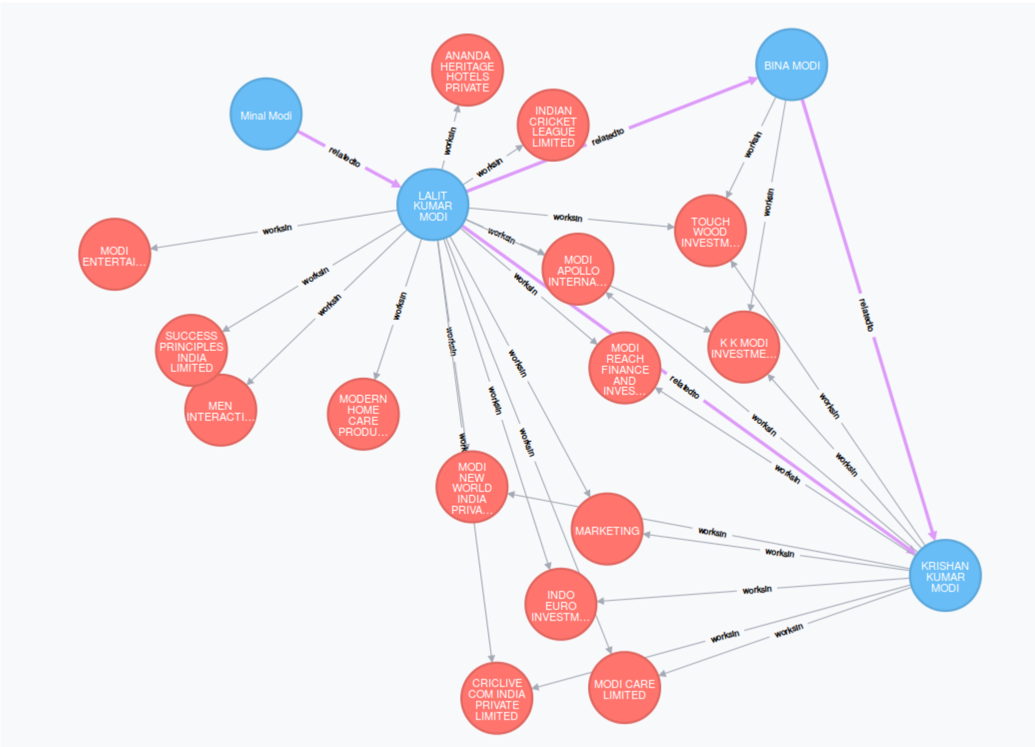


Figure 4.10: Lalit Modi of Modi dynasty

4.5.2 Mallyas

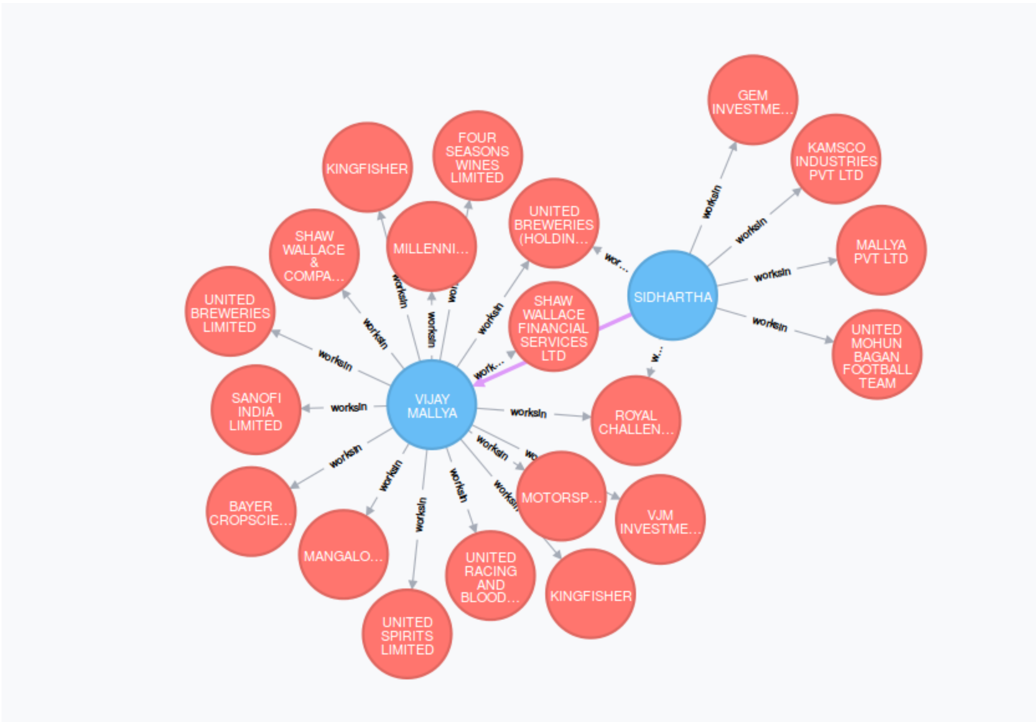


Figure 4.11: Vijay Mallya's empire

4.6 Arts, sports and commerce

4.6.1 Bachchans, Nandas, & Kapoors

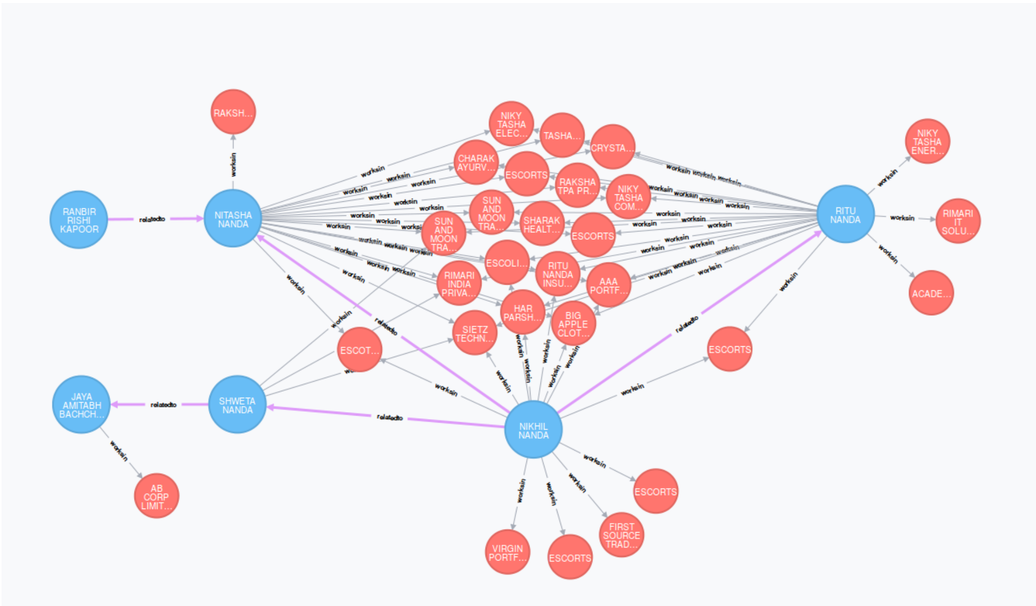


Figure 4.12: One of the best examples of big houses marrying richer/bigger houses

4.6.2 King's XI Punjab and the boyfriend connection

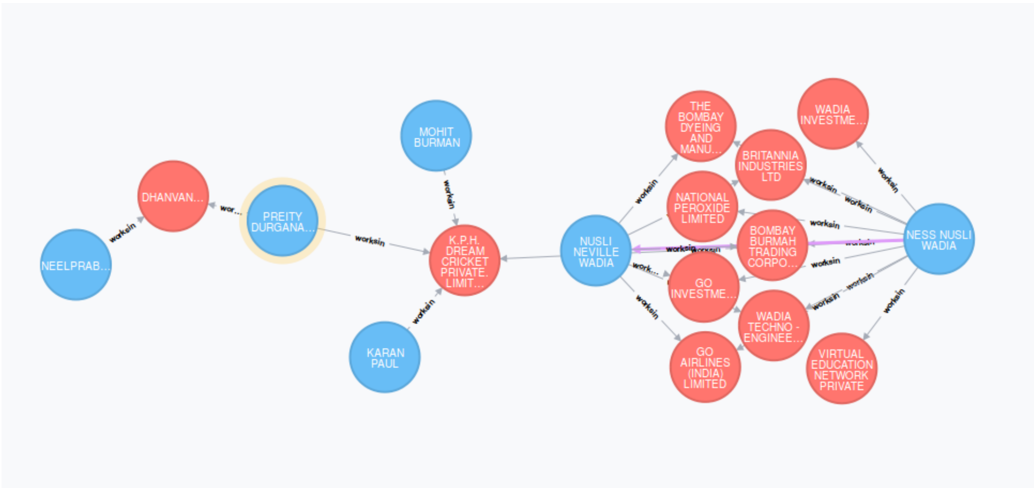


Figure 4.13: Preity Zinta, holding a directorship in KXP, with the then-boyfriend Ness Wadia

4.6.3 Sourav Ganguly a.k.a Dada

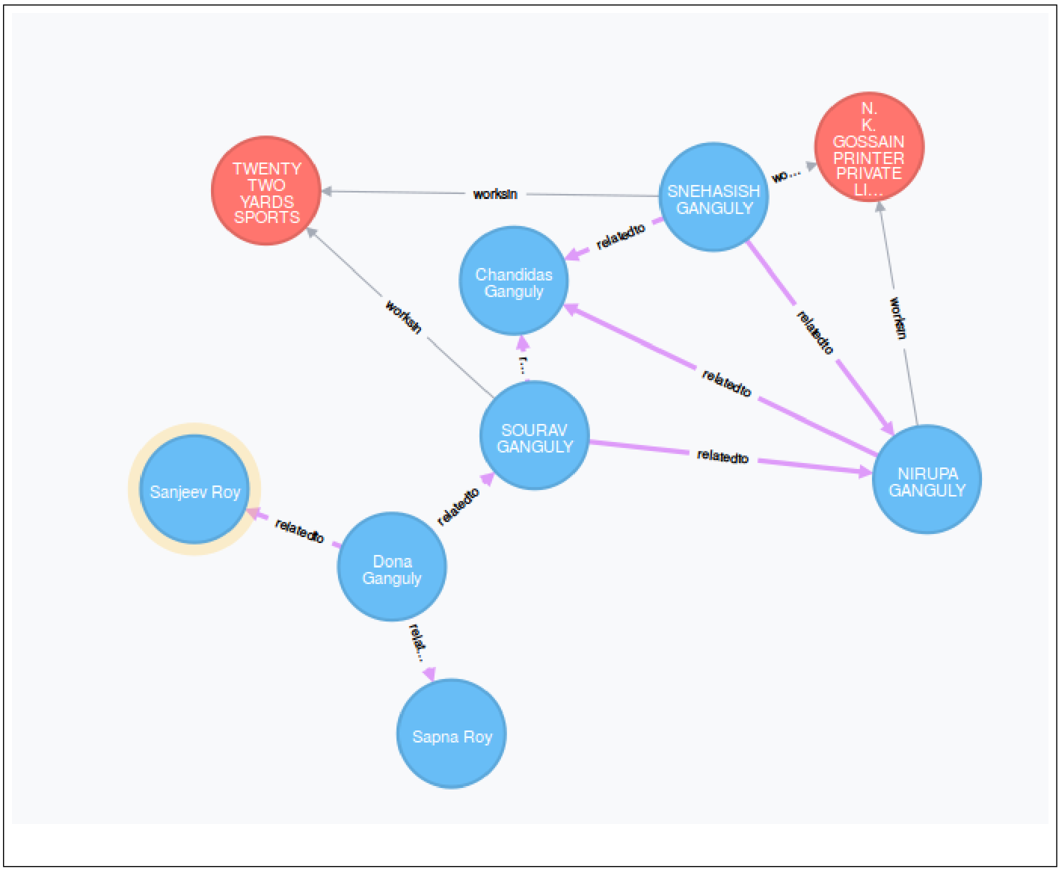


Figure 4.14: Sourav Ganguly and his company; His father is one of the richest men in Kolkata

4.7 Politics and Corporate

There are numerous well known examples like Naveen Jindal and Jaydev Galla that recur time and gain in news. Here we mention some more.

4.7.1 Gandhis and Vadra

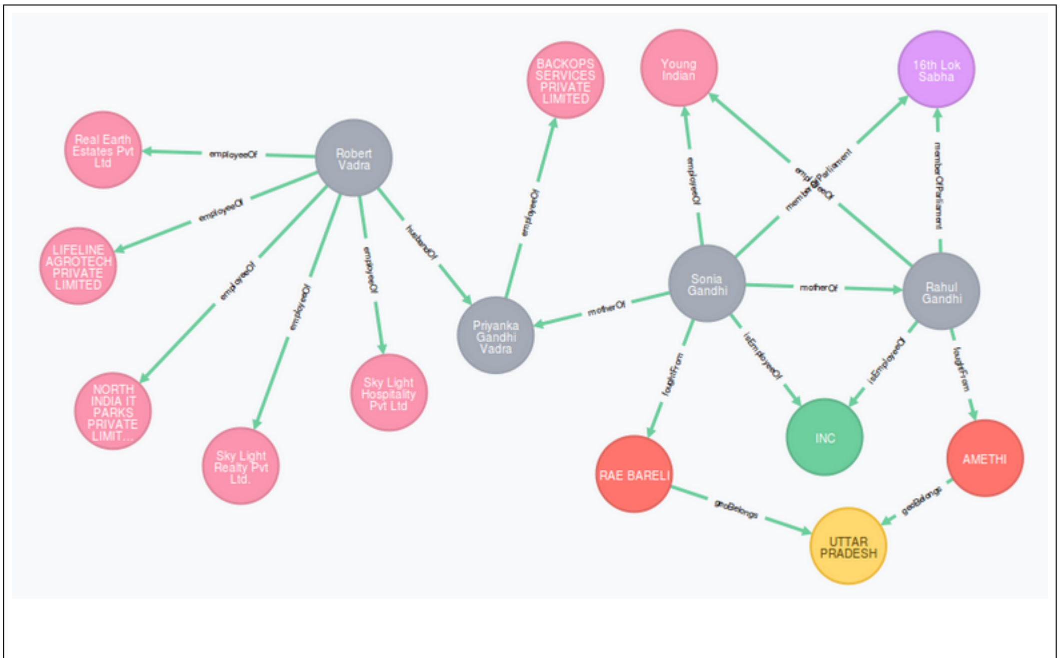


Figure 4.15: Gandhis and Son-in-law Vadra

4.7.2 Jayant Sinha and his business-cum-political family

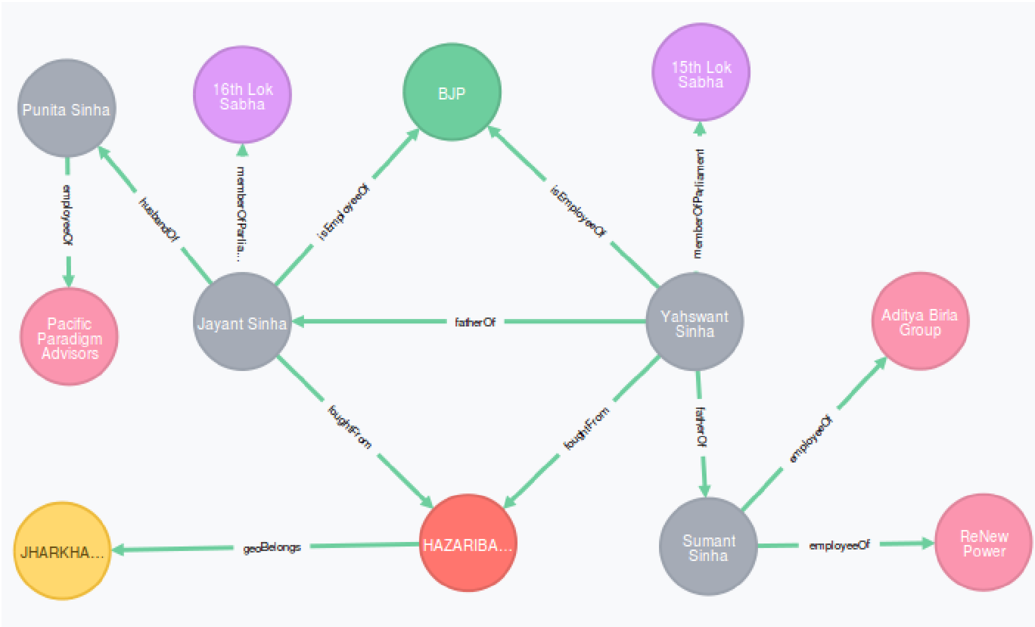


Figure 4.16: Jayant Sinha and Family

4.7.3 Kamal Nath and Moser Baer

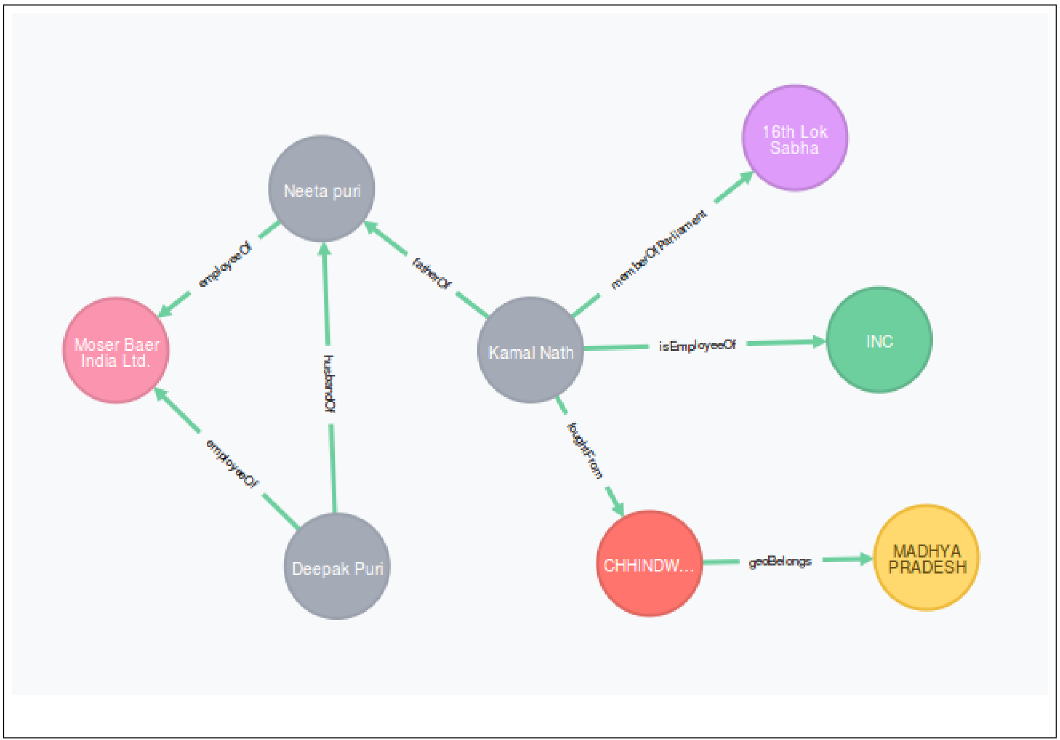


Figure 4.17: Kamal Nath and family

Chapter 5

Conclusion and Future Work

5.1 Conclusion

To summarize from the results of our queries in the graph, it is seen that the points we claimed previously are most of the times true. Clear connections do exist between corporate and political spheres in some cases as seen in the networks of Jayant Sinha and Kamal Nath. Moreover the clusters of Birlas and Ambanis suggest that the control and distribution of power is prevalent mostly in close family ties.

The work also showed the challenges of collecting and processing data from different sources especially in the Indian context. The absence of proper digitization of data, adherence to uniform formats and the absence of good interface for displaying data all added to the difficulty.

As a solution our system provides a good riddance from all above problems to become a common point for connected data access in the Indian context.

5.2 Future work

We have tried to start a process of building a system which in the long run will help the Indian society. Our best efforts were to cover as many functionalities as possible. Yet, many other problems and features are still left to be tackled.

- **Scaling up/out** - The designed system works fine and smoothly for the current amount of data. But question remains how to handle the data storage and processing when more data is pushed? Practically scaling can be done vertically by adding more powerful servers CPU or horizontally by the use of distributed databases.

- **Interactive Visualizations** - The visualizations existing presently in the system are very basic with simple interactions. More features can be added to help analysts, journalists to get more insights out of the data. Functionalities should be present to enable people to annotate a visualization, download it in different file formats, interact with it to reveal interesting patterns.

- **Better Query Engine** - Side-by-side with the visualization lies the query engine. As of now it only supports cypher queries as in *Neo4j*. But provisions can be made to add UI elements in a way such that users can form queries with little or no knowledge of cypher.

- **Inference Engine** - Till now it is the humans who are making analysis through the help of the knowledge base and visualizations. But we believe the system can be extended to let the machines draw conclusions from the same. Thinking in terms of Expert Systems and Symbolic AI, the knowledge base can be seen as a set of predicates (entities) and implications (relations). With the help of logic programming, one can write bots to infer automatically any implied relationships.

- **Better analysis of Social Networks** - The graph can be used to perform in-depth social network analysis. Factors such as centrality of the graph can tell about the important entities present. Similarly, weak ties, size of clusters in network can reveal information about lobbying. Also a comparison between our graph with a random graph may lead to discovery of interesting social network patterns.

- **Other use cases** - Several other use cases can be incorporated and analyzed with the data. Data about the IAS officers might show the possible relationships between the bureaucracy and the elite. Data about financial contributions of individuals or institutions might show the affiliations of big players with other entities. Moreover, the ownership data of different media houses can be used to explain their possible content.

Bibliography

[1] Apache lucene - apache lucene core. https://lucene.apache.org/core/.

[2] Apache solr. http://lucene.apache.org/solr/.

[3] Apache solr 6.1.0 documentation. http://lucene.apache.org/solr/6_1_0/index.html.

[4] Capitaline databases. https://www.capitaline.com/SiteFrame.aspx?id=1.

[5] Category:business families of india - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Category:Business_families_of_India.

[6] datamade/dedupe: A python library for accurate and scalable data de-duplication and entity-resolution. https://github.com/datamade/dedupe.

[7] google/cayley: An open-source graph database. https://github.com/google/cayley.

[8] Ims notes. http://www.people.vcu.edu/~lparker/IMS.html.

[9] Mariadb.org - ensuring continuity and open collaboration. https://mariadb.org/.

[10] The neo4j developer manual v3.0. https://neo4j.com/docs/developer-manual/current/.

[11] Neo4j: The world's leading graph database. https://neo4j.com/.

[12] Orientdb - orientdb distributed graph database. http://orientdb.com/orientdb/.

[13] Political families of india - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Political_families_of_India.

[14] What is a graph database? a property graph model intro. https://neo4j.com/developer/graph-database/.

[15] Association of democratic reforms - non-governmental organization for electoral and political reforms. http://www.adrindia.org/. 2015.

[16] Forbes - 147 companies that control everything. http://bit.ly/1O4BIgZ. 2015.

[17] Littlesis - profiling the powers that be. http://littlesis.org/. 2015.

[18] Ministry of corporate affairs - gateway to all services, guidance, and other corporate affairs related information in india. http://www.mca.gov.in/. 2015.

[19] Myneta - criminal and financial data on politicians. http://myneta.info/. 2015.

[20] Newslaundry - who owns your media? http://bit.ly/1eY5Bj2. 2015.

[21] Poderopedia - collaborative platform that helps understand the relationships among important people, companies and organizations for chile. http://www.poderopedia.org/. 2015.

[22] Sinha family tree. http://bit.ly/1O4BIgZ. 2015.

[23] Halbert L Dunn. Record linkage. *American Journal of Public Health and the Nations Health*, 36(12):1412–1416, 1946.

[24] Patrick French. *India: A portrait*. Vintage, 2011.

[25] Devesh Kapur and Milan Vaishnav. Quid pro quo: Builders, politicians, and election finance in india. *Center for Global Development Working Paper*, (276), 2011.

[26] Donald E. Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, 2 edition, 5 1998.

[27] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[28] C Wright Mills. *The power elite.* Oxford University Press, 1999.

[29] Lawrence Philips. Hanging on the meta-phone. *Computer Language*, 7(12 (December)), 1990.

[30] Thomas Piketty. Capital in the 21st century. *Cambridge: Harvard Uni*, 2014.

[31] Trilochan Sastry. Towards decriminalization of elections and politics. *Economic and Political Weekly*, 4, 2014.

[32] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill Education, 6 edition, 1 2010.

[33] Milan Vaishnav. *The Merits of Money and Muscle: Essays on Criminality, Elections and Democracy in India.* PhD thesis, Columbia University, 2012.

[34] William E Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.