An Efficient Schema Extraction Technique for Graph Databases

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Manpreet Singh

(Roll No. 801332014)

Under the supervision of:

Ms. Karamjit Kaur

Lecturer

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

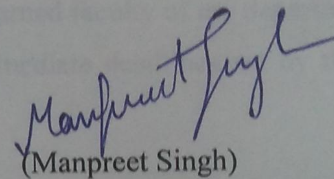
PATIALA – 147004

July 2015

CERTIFICATE

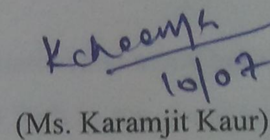
I hereby certify that the work which is being presented in the thesis entitled, "*An Efficient Schema Extraction Technique for Graph Databases*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Karamjit Kaur* and refers other researcher's work which are duly listed in the bibliography section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.



(Manpreet Singh)

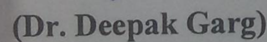
This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

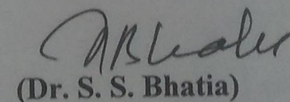


(Ms. Karamjit Kaur)

Lecturer, CSED

Countersigned by





(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

(Dr. S. S. Bhatia)

Dean (Academic Affairs)

Thapar University

Patiala

ACKNOWLEDGEMENTS

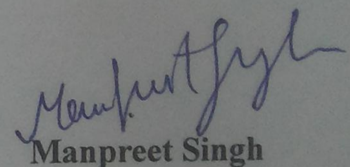
I am highly grateful for the assistance of all the individuals who spared their valuable time to help me complete my thesis work in time. Foremost I would like to thank my supervisor Ms. Karamjit Kaur for her constant guidance, in-depth knowledge, immense patience, invaluable time, encouraging words and worthy suggestions that helped me in completing my work. Under her guidance, I explored new areas of research. Her practical work approach helped me expand my horizon of research perspective. She has always been a moral support and true guide throughout my research period.

I am also thankful to Dr. Deepak Garg, Head of CSED, and the learned faculty of my department Computer Science and Engineering for their support. The intermediate deadlines set by them helped me work more efficiently.

I thank the librarian and all the lab assistants of our department who provided me with the basic resources that I needed for my research. I am indebted to my prestigious institute Thapar University, for providing me a learning and progressive environment.

My special thanks to all the friends for their endless support, appreciation and moral boost at times of need. I owe a lot to my family for their emotional support and faith in me, for providing me with an opportunity to be a part of this research programme.

Last but not the least I thank Almighty for providing me enough inner strength to pursue my goals and accomplish my work.



Manpreet Singh

ABSTRACT

Predominantly Relational Database Management System (RDBMS) is the major storage system deployed in health-care information systems. Even though it is widely used and remarkably mature, data with high degree of relationships levies a high performance toll on RDBMS. Heavy annotation of health-care data with relationships makes its storage suitable for specialized data models like graph databases or other considerably young NoSQL datastores. Each storage system has its own pros ans cons, the new NoSQL datastores cannot be considered the successors of the hugely established relational model. But storage mechanisms' true performance can be harnessed by using these models side by side, employing each model's specialty to store partial data in individual model. This approach needs connectivity and coordination among adopted datastores for precise data integration. Knowledge of schema of each participating datastore is essential for data integration, which the NoSQL datastores do not possess.

In this proposal an approach to extract schema from a running graph datastore (namely Neo4j) and its storage in a universal format (Datalog) has been proposed. The methodology employs universal approach for schema extraction by exploiting graph data exported into XML (GraphML) which is supported by most of the NoSQL datastores, hence making the approach generic. Further, algorithms are presented on handling and formulating user queries on the graph data utilizing the extracted schema. The procedure has been illustrated with the help of a medical database - EVD (Ebola Virus Disease). The proposed work is efficiently extracting schema from Neo4j but in order to include other database classes schema extraction technique will have to be slightly modified.

Contents

Certificate i

Acknowledgements ii

Abstract iii

Contents iv

List of Figures vi

List of Tables vii

1 Introduction 1
- 1.1 NoSQL Datastores 1
  - 1.1.1 Document oriented datastores 3
  - 1.1.2 Columnar datastores 4
  - 1.1.3 Key-Value datastores 5
  - 1.1.4 Graph based datastores 6
- 1.2 Comparison of NoSQL classes and RDBMS 7
- 1.3 Datalog 8
- 1.4 Thesis organization 10

2 Literature Survey 11
- 2.1 Graph datastores 12
  - 2.1.1 Properties and Usage 13
  - 2.1.2 Neo4j 14
- 2.2 Schema representation 16
  - 2.2.1 Ontologies & Protégé 16
  - 2.2.2 Knowledge base 18
  - 2.2.3 Prolog 19
  - 2.2.4 Datalog 20
- 2.3 Data Integration 21

3 Problem Statement 24

4 Graph Datastores 25
- 4.1 Features of Graph datastores 26
  - 4.1.1 Scalability and Flexibility 26
  - 4.1.2 Transactional properties 27

4.1.3 CAP Theorem 28
4.1.4 BASE Property 29
4.1.5 ACID vs BASE 29
4.1.6 Why Graph Datastores? 30
4.1.7 Applications of Graph datastores 30
4.2 Neo4j 32

5 Datalog **34**
5.1 First Order Logic (FOL) 35
5.2 Deductive databases 37
5.2.1 Datalog 37
5.3 Relational Algebra and Datalog 39
5.3.1 Datalog to RA 40

6 Implementation **45**
6.1 Tools and Environment 46
6.1.1 Python 46
6.1.2 PyDev 48
6.1.3 pyDatalog 50
6.1.4 Neoclipse 50
6.1.5 Implementation Environment 53
6.2 Proposed Algorithms 53

7 Running Example and Results **59**

8 Conclusion and Future Scope **69**

Bibliography **70**

List of Publications **75**

YouTube Video Link **76**

Reflective Diary **77**

List of Figures

1.1 Simplistic representation of data in Document based datastore . . 4
1.2 Simple columnar datastore representation 5
1.3 Representation of data in KVS datastore 6
1.4 Simplistic representation of data in graph datastore 7

2.1 Simplified patient record representation in graph datastore 12
2.2 Simplified patient record representation using Neo4j 15
2.3 Creating nodes and edges in Neo4j using cypher 15
2.4 Therapy ontology 17
2.5 Multi-store integration architecture 22

4.1 Simple graph representation 26
4.2 CAP Theorem 28

6.1 Python 2.7 snapshot in Linux 47
6.2 IDLE using Python 3.4 48
6.3 PyDev screenshot from Ubuntu 49
6.4 Simple pyDatalog program 51
6.5 Neo4j default visualizer 52
6.6 Neoclipse visualizer 52
6.7 Process followed for schema extraction 54

7.1 EVD graph (Snapshot from Neo4j console) 60
7.2 Screenshot of EVD database (partial) used 62
7.3 Real-time datalog facts, rules and queries for schema. 64
7.4 Node creation in pyDatalog 65
7.5 Edge creation in pyDatalog 66
7.6 Screenshot of Datalog rules used by Algorithms 67
7.7 Queries: To get traversal path between two nodes 67
7.8 Results: Traversal paths returned by the Algorithms, between two nodes 68

List of Tables

1.1 Comparison of NoSQL classes and RDBMS 8
1.2 Operations in Relational ALgebra and Datalog 9

4.1 Cypher clauses 33

5.1 Doctor Table in RDBMS 40
5.2 Specialization Table in RDBMS 40
5.3 Doctor-Specialization Table in RDBMS 40

Chapter 1

Introduction

Data is said to be an assimilation of statistical figures or raw facts, collected for analysis or reference. It is the atomic entity that is yet to be processed to produce some meaningful information. This data is often used to portray the ecosystem around us in different ways and thus helps us in augmenting reality into digitally storable bits and bytes. Database is an organized way of arranging this raw data in such a manner that it can be easily accessed, maintained and altered. Typically related data is stored in a database based on some relation enacting some real-life aspects. Various specifically designed softwares or applications have been created today that take raw data either from end users or from other similar databases or applications. These softwares or applications are often called Database Management Systems (DBMS). DBMSs help in creating, altering, querying and administering these databases. Various well-known DBMSs are MySQL [1], Oracle [2], PostgreSQL [3], CouchDB [4], Redis [5], etc.

1.1 NoSQL Datastores

With the advent of Web 2.0, a term given by Darcy DiNucci in 1999 the whole internet ecosystem was revolutionized by the introduction of more dynamic and active data instead of static or passive data [6]. The social networking websites, blogs, video sharing websites, web applications, etc. provided a new look to the internet and changed the way people interacted with it. All this and the successful and widely accepted launch of HTTP during that period led to an overflow of information on the internet. It was soon noticed that this data could not comply with the conventional schema oriented approach and the unstructured data produced by the internet could not fit into the tabular format of the RDBMS. The issue that arises due to this increase in unstructured data is how to handle this volume growth efficiently and in the most cost effective

way possible. This is termed as scalability. There are two approaches to attain scalability viz. vertical scalability and horizontal scalability. Vertical scalability or scaling up means enhancing a single system's resources to gain performance. It is easier to implement but not optimal. Horizontal scalability or scaling out refers to increasing the number of nodes in parallel or in support of the current system. Traditional RDBMS supported vertical scalability more whereas non-relational databases can easily accommodate to the changing data needs by using horizontal scaling.

This voluminous and unstructured data often known as "Big Data" due to their complexity and largeness. Big data is generally defined as "3Vs model", where these Vs stand for Volume, Velocity and Variety of data [7]. By volume we mean the vast amount of data that is being generated on the internet everyday. Velocity refers to the rate at which this data is being generated. Variety depicts the various formats of digital data roaming on the internet varying from simple texts files of few kilobytes in size to videos of few gigabytes in size. Taking this schema less unstructured data and the growth rate into consideration various data processing oriented firms started to tilt away from the traditional tabular approach and in search for a perfect database that could befit their data storage needs, gave birth to various non-relational databases. In 2006, Google came up with its well-known "BigTable", a column-oriented database and laid the founding stone for the innovations and research in this area [8]. Later on following Google's footprints Amazon in 2007 came up with "Dynamo", a key-value based data store. [9] Later on various such databases supporting the non-relational approach were introduced in the market and our further discussion will be based on these databases only.

NoSQL database systems are the latest and new generation systems in the field of non-relational databases that help design and develop those applications that require management of such data which is not well suited for traditional RDBMS. NoSQL term in normal use implies to or covers all the non-relational databases. It stands for "Not only SQL" but there is a debate on the name as this tends to ignore the key features like non-schema, non-ACID databases.

These databases maybe quite young but they have shown their capability in this short premise of time. More than 50 NoSQL databases with varying characteristics and properties have already been implemented. As discussed earlier this concept was picked up by search engine giant Google in 2006 which led to the creation of "BigTable" data store based on the column oriented approach. Apart from the search engine application other services like Google+, YouTube, Gmail, Google Drive, Google Maps, etc. by Google started gaining attraction and thereby fetching a lot of users. This lead to eruption of vast amount of heterogeneous and unstructured data at Google data stores. They could not handle this kind of varied data with the help of traditional RDBMS. Moreover, with the coverage of more and more locations the distributiveness of their data stores started increasing. Increase in data wanted the system to implement scalability in a flawless manner. To cater to all these problems they included map reduce, distributed file system, column oriented storage techniques for their data store.

After sometime when Google and Dynamo revealed their approaches and the performance gain they have achieved others also started looking towards such approaches. In present day scenario Facebook, Twitter, Yahoo, eBay, etc. all are using NoSQL approaches to store their data efficiently. A lot of NoSQL databases exist today, some of them are listed here: MongoDB [10], Redis [5], CouchDB [4], Cassandra [11], Neo4j [12], BigTable [8], Dynamo [9], etc.

Various NoSQL data model classes exist to befit the data storage needs of all kinds of vivid users and developers. Some of these classes have been presented in the following section.

1.1.1 Document oriented datastores

In this model, documents are used to store data. Semi-structured data is well managed by this model. CouchDB and RavenDB are example databases of document data model that use JSON format to store data whereas MongoDB, another database based on this model, uses BSON format to store data [10]. JSON stands for JavaScript Object Notation. It emerged as an alternative to XML and is used for transmission of data between a server and web applications. The

1.1.2 NoSQL

NoSQL, an open source database, name given on basis that we should not only used SQL databases and name given by Carlo Strozzi [5]. Then NoREL name is given to it because has no relation with relational databases. It does not follow any rules of RDBMS. NoSQL can work with any type of data *i.e.* structured, unstructured or semi-structured, *i.e.* data in the formats of pdf, html, images which is very difficult to handle with relational databases. It has flexible and dynamic schema, no need to define its schema prior.

The main reason of popularity of NoSQL databases is Scaling, means handling more data with increasing resources according to need of applications.

Example: Initially NoSQL concept is implemented by Google. Firstly it has only one search engine as with time it launched so many applications like google maps, google+, Gmail *etc.* It has now too much data which cannot be handled by RDBMS and also data is geographically dispersed in unstructured form. There is a need of a data store to handle this data which has no relation with relational database. Also to include parallelism need many resources *i.e.* need of scalability [6], so they include distributed file system, map reduce, column oriented data store which is popular as Big Table.

After inspired by google, Amazon has also found a non-relational database known as Dynamo which is eventually consistent and easy to use. Then Facebook, twitter, yahoo, eBay companies are also used these non-relational databases. There are many NoSQL databases exist in this world [7]. Some of them are MongoDB[4], CouchDB[5], Cassandra[6], Redis[7], Neo4j[8], Hadoop[9], Big Table [8], Dynamo [9] etc.

CAP THEOREM

Traditionally data is placed at one centralized place but now a day's data has crossed its limit so cannot be handled at one place, so distribute to multiple sites is needed. In distributed system still ACID properties can be achieved but so many applications do not want any type of consistency immediate after any operation. From this idea CAP theorem evaluates. CAP stands for **Consistency, Availability and Partial tolerance**.

---

[4] http://www.tutorialspoint.com/mongodb/
[5] http://guide.couchdb.org/
[6] http://cassandra.apache.org/
[7] http://redis.io/
[8] https://Neo4j.com
[9] https://hadoop.apache.org/

The other name of this theorem is ***two out of three*** [10] means at a time only two properties out of these three can be achieved.

If data is centralized means not partitioned then it means availability and consistency property can be easily achieved. But if data is partitioned into many sites then either consistency or availability can be achieved. In NoSQL those applications can be run which don't want ease of use and consistency both at a time.

Taking an example of updating status in Facebook. In this no need of any type of consistency and also no need that our whole servers are available because it may be happens that it is available to some people and later on it may be show to other people also [10]. The three basic parameters of CAP theorem are shown in Fig 1.3.

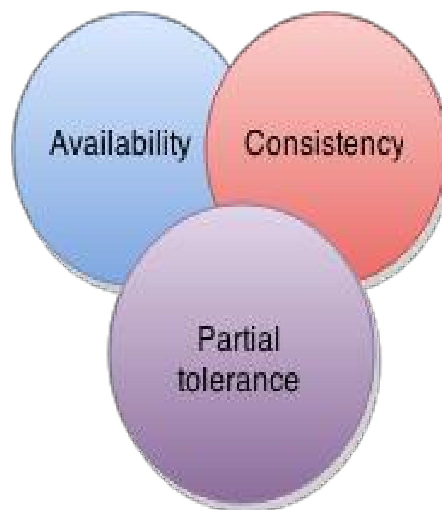


Fig 1.3: CAP Theorem

BASE PROPERTIES

BASE stands for **Basically Available, Soft state and eventually consistent** [11]. NoSQL has BASE properties based on CAP theorem. Based upon that BASE properties it is works in a manner that it may be eventually consistent having soft state when it is basically available.

- ***Basically Available:*** When data is stored at one site. If that site is down, data is totally unavailable but when data is distributed to many sites, if one or two sites down then whole data is not unavailable. It may be for some short time that some part of data becomes unavailable. So, data is now partially unavailable.

- ***Soft State:*** In BASE consistency is not primarily need of any applications which is hard requirement in ACID. So it leads dynamic designing to application developers. Soft state means state of system can be changed without giving any input.
- ***Eventually Consistent:*** Like in ACID properties they enforce consistency immediately after commit the transaction, in this BASE properties they guarantees consistency in some undefined future time, not immediately after the operations. It can be consistent after some time, no guarantee immediately after the operation.

Basic differences between ACID and BASE properties are explained in Fig 1.4.

ACID	BASE
It stands for Atomicity, Consistency, Isolation and Durability	It stands for Basically Available, Soft state and Eventually consistent
These are pessimistic that means it guarantees that consistency should be achieved at the end of any operation.	These are optimistic means that it provides no guarantees that consistency should be achieved at the end of any operation.
Basic requirement in RDBMS	Basic requirement in NoSQL
These are complex to implement all properties	These are simple to implement all properties
Provides reliability	Provides no reliability

Fig 1.4 Acid vs Base

In Section 1.2 why NoSQL are very much popular in front of relational databases are going to be discussed, what are the factors behind their popularity.

1.2 NoSQL Drivers

By the time it has reached a level where data is unable to handle process and store. Hence come across concerns like security issues, storage, authorization and lots more. Since traditional databases cannot be used for data storage as data is much unstructured and in distinctive formats [12]. There is no need to define schema in prior. Some of factors due to which NoSQL becomes so popular are explained as follows.

1. Big data

Now a days a large amount of data[10] and information in systems that has to handle. To handle this huge amount of data we require a room like structure where we can just simply keep our data without any organized manner. So NoSQL here acts like as a room for us to handle data in denormalized form. Example Facebook, Google have huge amount of data, they require a big store to keep this data so they are using NoSQL databases.

2. Scalability

When data was in small amount there is no need of scalability[11]. As data is rising at exponential rate so need to disperse data to many sites and also need more resources to handle the data. So scaling is needed here. There are two types of scaling one is vertical scaling and another is horizontal scaling [13]. In vertical scaling CPU resources to increase the capacity of memory, processors speed, to increase the parallelism can be added and in horizontal scaling whole computers acts as servers can be added to increase the computation power.

3. Availability

As in NoSQL data is partitioned into many sites. If one site is down at that time but other sites are available to continue the operation. But in centralized system if site is failed then whole data is unavailable.

4. Schema-less

As in relational database schema is needed to specify in advance to perform complex operations like join, normalization but here in NoSQL no need to specify the schema in advance. So NoSQL is very flexible and in this no operations like normalization and join operations needed to be performed.

5. Semi structured/Unstructured

As data is available in unstructured form so it is very complicated to map data to RDBMS. Even it is very complex with ER model. But in NoSQL unstructured type of data can be easily handled, no need of mapping even.

---

[10] http://www.oracle.com/bin-data/index.html

[11] https://www.mongodb.com/mongodb-scale

6. Aggregation in a cluster:

In RDBMS clusters of different types having different information is present but here in this need to collect the whole information from sites and then aggregate them into one cluster so that retrieval of information performs easier.

After understanding basic needs of use to non-relational databases, data in different forms can be represented that are explained in following section.

1.3 Data Model Representation

Consider a case where large amount of data of an employees, want to store personal details like name, surname, also want to store their residence details like city, state where he lives and also want to store their order details what they want to purchase. Every employee also provided a unique id through which he or she is identified. Now explaining in following sections that how this information can be represented in different models.

1.3.1 RDBMS

Database Management System is software which is used to process information and manage operations like insertion, deletion, updation. RDBMS[12] shows the relationship between tables and rows. In this data is store in the form of rows in a table. If we have a large amount of data we need different tables and if we want to execute a query we need joining between these tables. Joining is very complex and time consuming tedious task. So to improve performance and time consumption we use NoSQL Models. Fig 1.5 describes how these personal, residence, order details can be stored in table by defining its schema in advance. RDBMS has a problem that if we have large amount of data we need multiple tables. To perform any query we need some operations as we have to perform join operation. This operation is very cumbersome. To perform any query we need a query language RDBMS uses a SQL query language.

[12] http://www.databasedir.com/what-is-rdbms/

Personal Details

id	Name	Surname
1	Mukesh	Garg
2	ramesh	garg

Residence Details

id	City	State
1	Patiala	Punjab
2	Patiala	Punjab

Order Details

id	item No.	item Name
1	12345	Car
2	56489	Motorcycle

Fig 1.5: RDBMS

1.3.2 Document Oriented Data-Store

Based upon key value pairs but having value is in the form of well-defined document. These documents are of many types some of may be XML, JSON or any other forms. But here in thesis assuming that documents are in form of JSON[13] java script notation. Documents may be embedded in other documents and they may be linked to another document which permits us to show one to one or one too many relationships. These type of documents are nested documents. Now describing advantages over RDBMS is no need of joining in this document store database[14], Also there is no need to fix the schema in advance and also it can perform aggregation very easily. Fig 1.6 explains the structure of document in JSON structure having key and value pair. To perform any operation in document oriented we need some query language which will be discussed in Section 1.4.

[13] http://json.org/
[14] https://www.mongodb.com/document-databases

```
{
"id":1
"Name":"Mukesh"
"Surname" : "Garg"
"City":"Patiala"
"State" : "Punjab"
"Order":{
"Item No.": "23"
"Item Name": "Car"
"Model No." :"12345"
}
```

Fig 1.6: Document oriented data store

It can work with denormalized data and semi structured data very easily. Indexing can be easily done with document stores [14]. Some of the popular document databases we have seen are MongoDB, Couch DB, Terrastore[15], Orient DB[16], and Raven DB[17].

1.3.3 Key Value based Data Store

By taking an inspiration from data structures concept in which to search an element with the complexity of order 1 hashing technique is used, there is implemented a database known as key value data store[18]. In this data store a table in which keys are stored and corresponding values are associated with them is needed. Values can be accessed only by referring its corresponding key [15]. Based upon keys aggregate into clusters and horizontally scaling can be performed very easily. Also the operations like insertion, searching, deleting, updating can be executed very easily. It is easy to use and very simple model. Figure 1.7 explains that how data is stored in key value pair in key value based data store.

---

[15] http://www.terrastore.com/
[16] http://orientdb.com/orientdb/
[17] http://ravendb.net/
[18] https://foundationdb.com/key-value-store

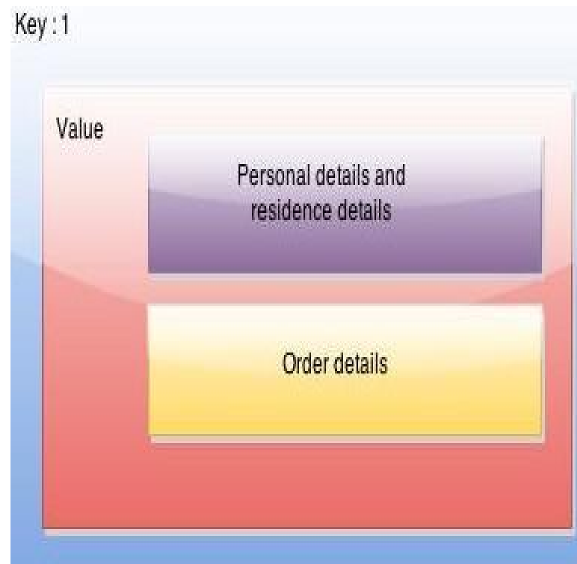


Fig 1.7: Key Value based Data Stores

Some of the popular key-value databases are Riak, Redis, Memcached and its flavors, Berkeley DB, HamsterDB, Amazon DynamoDB, Project Voldemort and Couchbase.

1.3.4 Column Oriented Data Store

As RDBMS contains row wise data but in NoSQL now there is a flexibility to store data in column wise[19] .With this scenario in which no schema fix in advance can be implemented. There is one row which contains multiple columns and these columns are different from other columns in different second row means every row contains different types of columns and different numbers of columns [16]. Fig 1.8 explains that how data is stored in the form of columns not in rows.

[19] http://www.slideshare.net/arangodb/introduction-to-column-oriented-databases

1	2	3
Mukesh	Ramesh	Rajesh
Garg	garg	Aggarwal
Patiala	Patiala	Patiala
Punjab	Punjab	Punjab
12345	56489	8456
car	Motorcycle	Scooty

Fig 1.8: Column Oriented Data Store

The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows. Every column has a key value pair associated with it means there is one key and equivalent there are multiple values stored in one column [16].So, column family store is basically used for read purposes. Some of the popular column family databases are Cassandra, HBase[20], Hypertable[21], and Amazon DynamoDB.

1.3.5 Graph based Data Store

Edges and nodes are basic two terms in graph based data store[22]. Here nodes are vertices and relationship between that nodes acts as edges. It is easily modeled to ER model of relational model where entities played role as vertices and relationships are edges. Every node and edge has properties [17]. Edges have directional significance. In NoSQL this is also using key value property in which every node has a key and values. Even relationships also have key and values. Nodes can have different types of relationships between them. It is used where highly relatedness between data. In RDBMS to find any relationships between two items, join operation need to be performed but in NoSQL no need for join operation. Fig 1.9 explains that how data is represented in graph based data store [18]. Some of the popular graph databases are Neo4J, Infinite Graph, OrientDB, or FlockDB.

[20] http://hbase.apache.org/
[21] http://hypertable.org/
[22] http://neo4j.com/developer/graph-database/

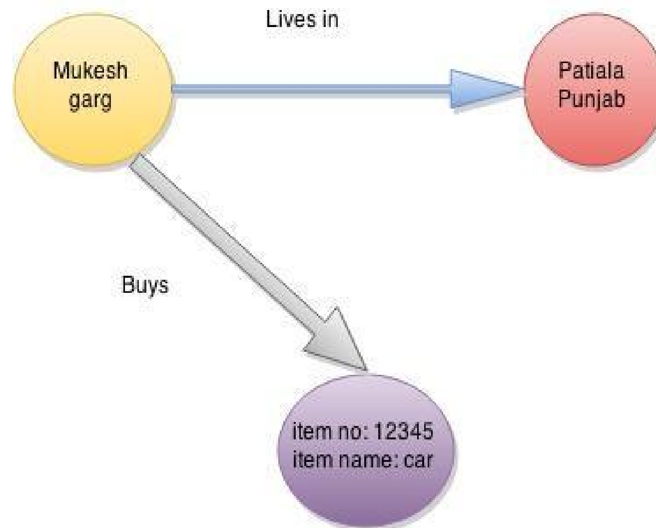


Fig 1.9: Graph based Data Store

1.4 MongoDB

Moving from relational to distributed systems and then to NoSQL databases as data is increasing at high speed and in unstructured manner, cannot depend upon only centralized systems, need to scale up and scale out as having denormalized data NoSQL is in great demand. Here document oriented store model of NoSQL is focused in this thesis. As there are many examples of documented oriented databases, MongoDB is going to be discussed in following section.

1.4.1 Background and Properties

Data is increasing so much fast rate, so cannot think about schema in advance. So need a schema less database that can be easily scalable and also follow the BASE properties. So MongoDB is document oriented store having so many properties. It is developed by **10gen** Company. It is originated from word **humongous**[23]. It is open source, free software has cheaper storage and used for web based applications. It is used in those where applications need updating, insertion. It is simply performed join and complex operations have dynamic schemas. It is written in C++ and supports ad hoc queries. It is used for load balancing means scaling can be done with MongoDB is very easily. It is based on document store, it contains JSON java script notation documents having semi structured data. It has APIs that can supported any programming languages that is why it can easily use with any programming language

[23] https://www.mongodb.org/

like python, java etc. here in this thesis for implementation Mongo version 2.0.2 is used. It has similarly query structure like SQL. It is basically in between SQL and NoSQL.

1.4.2 Data Types Used in MongoDB

A database consists of collections and further that collection consists of documents. Each document is a structured document that has, a complex value, a set of attribute-value pairs, which can comprise simple values, lists, and even nested documents. Documents are schema-less, that is, each document can have its own attributes, can be created at runtime. MongoDB documents are based on JSON. Values constituting documents can be of the following types:

1.4.2.1 Data Types

String: a string value (e.g. "abc", "river"),

Number: a numeric value (e.g. 1234),

Boolean: a true/false value (e.g. true, false),

Null: a non-existent value.

1.4.2.2 Arrays

In these arrays contains heterogeneous types of data which is contrast to array in C, C++.

Things: ['abc','def', 1, true]

1.4.3 MongoDB Queries

Basic MongoDB queries[24] are explained in following sub sections. Basic queries include insert, serach, update, group by, order by, delete *etc.*

1.4.3.1 Insertion

In this insert of a document in a collection is explained. Here a document having personal details, educational details and other details of a doctor is inserted in doctor collection. The query to insert a document is written in following table.

[24] http://docs.mongodb.org/manual/reference/operator/query/

MongoDB	db.doctor.insert({"_id" :3 ,"offics_location" : "Patiala","personal_details":{"fname":"Anand", "mname" : "kumar" ,lname":"sehgal"} "educational_details":{"medical_school": "Rajindera college" }})

The snapshot of how to write a query on mongo console is shown in Fig 1.10.

```
> db.doctor.insert({"_id":45,"offics_location":"patiala","Available_day":["mon",
"wed","fri"],"personal_details":{"fname": "Aulakh", "mname": "kumar", "gender" :
 "male","address":"modeltown phase1","city": "patiala","zipcode" : "147008" , "s
tate" : "punjab", "country": "india",  "nationality" : "indian"} ,"educational_d
etails":{"medical_school":"Rajindera college","internship":"dmc amritsar","speci
ality":["consultant Physician"]},"other_details":{"work_experience":"5 years","h
ospital_joined_date":"2008-01-12"}})
> db.doctor.update({"_id":45},{"$set":{"personal_details.fname":"Anand"}})
> db.doctor.find({"_id":45}).forEach(printjson)
{
        "Available_day" : [
                "mon",
                "wed",
                "fri"
        ],
        "_id" : 45,
        "educational_details" : {
                "medical_school" : "Rajindera college",
                "internship" : "dmc amritsar",
                "speciality" : [
                        "consultant Physician"
                ]
        },
        "offics_location" : "patiala",
```

Fig 1.10: Insertion of data

1.4.3.2 Searching

Here we are going to describe that how we can find any document in a collection in MongoDB The query written in a table is finding documents from doctor collection.

MongoDB	db. doctor. find();

In Fig 1.11 snapshot of mongo console explains how to write a query of basic search and print in JSON format. Here forEach function is used to print the results in JSON format.

```
> db.doctor.find({"_id":39}).forEach(printjson)
{
        "_id" : 39,
        "offics_location" : "patiala",
        "Available_day" : [
                "mon",
                "tues",
                "wed"
        ],
        "personal_details" : {
                "fname" : "Aulakh",
                "mname" : "kumar",
                "gender" : "male",
                "address" : "ModelTown phase1",
                "city" : "patiala",
                "state" : "punjab",
                "country" : "india"
        },
        "educational_details" : {
                "medical_school" : "Rajindra_college",
                "internship" : "dmc amritsar",
                "speciality" : [
                        "consultant_physician"
                ]
```

Fig 1.11: Searching in MongoDB

1.4.3.3 Updation

In SQL we can update only those items which are in the document but in MongoDB there is a command upsertion that if document is not there in the collection which we want to update then it inserts that document firstly then update accordingly.

MongoDB	db. doctor. update({ age: {$gt:25} , { $set : {“status”:”A” } } ,{multi:true});

In Fig 1.12 snapshot of mongo console explains how to write a query of update a document. In this command $set is used to update document that explains which column field of a document is to be updated.

```
> db.doctor.insert({"_id":45,"offics_location":"patiala","Available_day":["mon",
"wed","fri"],"personal_details":{"fname": "Aulakh", "mname": "kumar", "gender" :
 "male","address":"modeltown phase1","city": "patiala","zipcode" : "147008" , "s
tate" : "punjab", "country": "india",  "nationality" : "indian"} ,"educational_d
etails":{"medical_school":"Rajindera college","internship":"dmc amritsar","speci
ality":["consultant Physician"]},"other_details":{"work_experience":"5 years","h
ospital_joined_date":"2008-01-12"}})
> db.doctor.update({"_id":45},{"$set":{"personal_details.fname":"Anand"}})
> db.doctor.find({"_id":45}).forEach(printjson)
{
        "Available_day" : [
                "mon",
                "wed",
                "fri"
        ],
        "_id" : 45,
        "educational_details" : {
                "medical_school" : "Rajindera college",
                "internship" : "dmc amritsar",
                "speciality" : [
                        "consultant Physician"
                ]
        },
        "offics_location" : "patiala",
```

Fig 1.12: Updation in MongoDB

1.4.3.4 Group by and order by

Here we are going to describe that how we can aggregate our data into one group and also if we want to retrieve our data according to some order either increasing or decreasing then query syntax is written below.

MongoDB	db.doctor. Aggregate ([$group: {_id:" $educational_details. medical_school",{addToSet: {name:"$personal_details.fname"} } }] , [{ $ sort : { experience : 1}}])

In Fig 1.13 snapshot of mongo console explains how to write a query of group of date. Here aggregate function is used to run group by command. The resulted generated by aggregate functions is always in JSON format.

```
> db.doctor.aggregate([{$group:{_id:"$educational_details.speciality",details:{$a
ddToSet:{office:"$offics_location",experience:"$other_details.work_experience",na
me:"$personal_details.fname"}}}}],[{$sort:{experience:1}}])
{
        "result" : [
                {
                        "_id" : null,
                        "details" : [
                                {
                                        "office" : "patiala",
                                        "name" : "Aulakh"
                                },
                                {
                                        "office" : "patiala",
                                        "experience" : "5 years",
                                        "name" : "anand"
                                }
                        ]
                },
                {
                        "_id" : [
                                "consultant Physician"
                        ],
                        "details" : [
```

Fig 1.13: Group by and Order by in MongoDB

1.4.3.5 Deletion

In this how to delete a document from a collection is explained. Remove function is used to delete a document from a collection. Command is written in following table. This command is deleting all the document from doctor collection who has status is equal to “A”.

MongoDB	db.doctor.remove({“status”:”A”})

How to delete a document from a collection whose id 4 is shown in Fig 1.14

```
                        "consultant Physician"
                ]
        },
        "offics_location" : "patiala",
        "other_details" : {
                "work_experience" : "5 years",
                "hospital_joined_date" : "2008-01-12"
        },
        "personal_details" : {
                "address" : "modeltown phase1",
                "city" : "patiala",
                "country" : "india",
                "fname" : "Anand",
                "gender" : "male",
                "mname" : "kumar",
                "nationality" : "indian",
                "state" : "punjab",
                "zipcode" : "147008"
        }
}
> db.doctor.remove({"_id":41})
> db.doctor.find({"_id":41})
>
```

Fig 1.14: Deletion in MongoDB

1.4.4 Map Reduce in MongoDB

As we have lot of data we cannot store on one database and also cannot run at one time. So, a framework or model which can run program parallel is needed, that framework is provided by MongoDB. MongoDB can run this map reduce model [10]. By this data is divided into machines to reduce workload. Traditionally have only serially programming, need to divide program so that it can processed concurrently. Map reduce[25] is inspired by LISP programming, it provides a framework for parallel computing. It is created by Google in 2004. It is programming model processes lots of data to produce other data. In this model we have set of key value pairs.

[25] http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Programmers need to specify two operations. One is Map and another function is reduce function.

Map(key, value) -> list (out_key, intermediate_value)

Reduce(out_key, list(intermediate_value))->list(out_value)

In this map function processes data and taken as key value pair and list out some type of intermediate data. Then that intermediate pass to reduce function and then after process and lists out output, or it combines intermediate values and produces a set of merged output values.

Main Functions in Map reduce:

- Map
- Partition
- Sort
- Merge
- Reduce

Divide work into some tasks so that can run parallel, they passed to Map function where this map function grabs the relevant data in the form of key value pair and write it to intermediate file. Then there is a partition function which helps us to assign tasks to corresponding reducers which reducer handles which key. Then sort function will sort all data based upon some criteria. Then reducer function processes data and merge/combine function will give the output to the file.

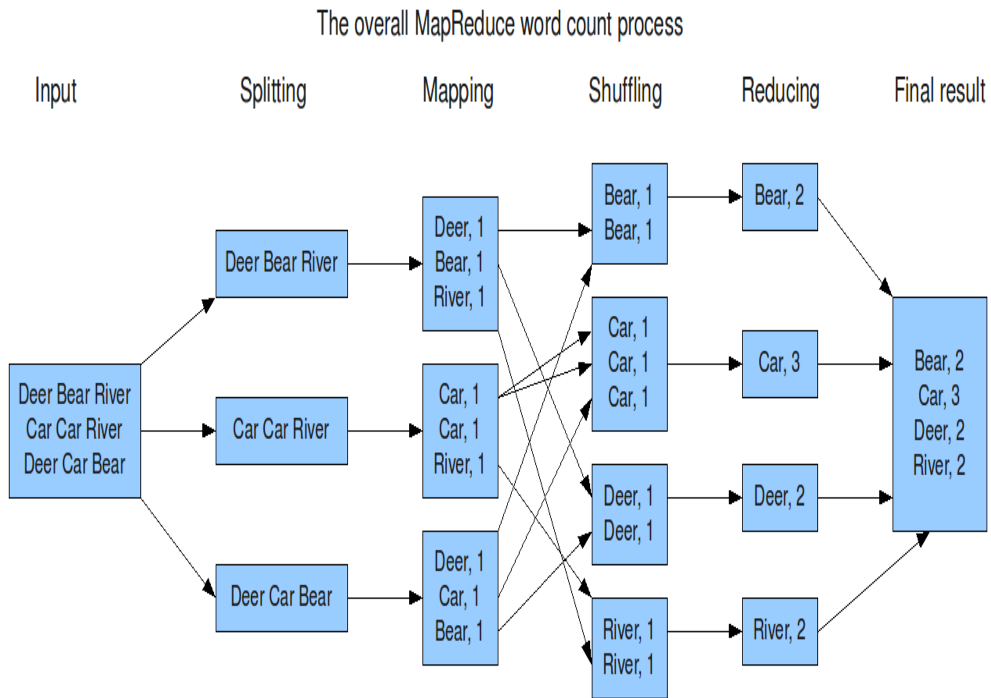


Fig 1.15: Map reduce in MongoDB

Here in this example, a word count is shown. A document is entered, initially it splits into 3 documents so that can run parallel and also these are independent tasks. This is called sharding[26] of document. Then these documents are passed to Map function where that processes data and count of occurrences of words and write it to intermediate files. Then with the use of shuffling function we combines the data which are related to one machines that are reducer machines and that update the results of words and process that data. Finally they write all data to the output file. This is basic structure of map reduce in MongoDB.

1.5 Challenges

Understanding representation of data in different formats, found that every model has its own query, own data model. But cannot store the entire data with only one non-

[26] http://docs.mongodb.org/manual/sharding/

relational database or any relational database. So spread data across different locations and in different data stores according to need of applications is needed. In the end then integration of that data is needed. For that data integration, knowledge of schema is the primarily need. Further discrepancy arrived with non-relational databases was its unstructured and dynamic view of schema. Here in this thesis, schema information of MongoDB database is fetched. As Mongo DB has dynamic structure of its schema, anyone can store any key. With the use of MongoDB, mongo Inspector and python tkinter, GUI based tool is implemented. Users and developers have not any proper view. To decrease that confusion, schema extractor is very helpful to them. With the addition of this feature, MongoDB as relational SQL database can be used.

1.6 Organization of thesis

As we have discussed the introduction about databases, that how moved to non-relational databases, what are the challenges found while using non-relational databases, that data is too large so cannot store in one database, need to spread data across different sites, and in different databases, for this data integration is needed. For that schema knowledge is must. So work till now has been done for extracting schema information, that is discussed in literature review in chapter2. Then what type of problems and what motivate us to do work on this schema extractor is discussed in chapter 3.Then in chapter 4 implementation of project, steps to be followed and used algorithm is explained. Then what are the outcomes of this implementation and results are explained in chapter 5. The last chapter regarding conclusion and future scope of the work.

Chapter 2

A Brief Review of Existing Schema Extractors

This chapter gives a brief review of some of existing visualization tools. Section 1 explains a Schema.js tool used for visualization and its output is also shown with commands. Section 2 describes variety used for analysis and visualization with its results and commands. In section 3 mongo Inspector using map reduce tool used for visualization with output with commands. Also In this chapter why Mongo Inspector is better than other tools are going to be explained and how it is used in implementation is also discussed.

As MongoDB is a non-relational database, as column oriented, it becomes more popular. But the problem arrived is that it has dynamic view of schema. And in data integration main need is schema analysis. Schema analysis is a brute force approach which involves looking at the data in order to infer an observed schema. Schema means define the structure about collections like which keys are stored in documents of collection, relationalships between collections. As in relational we have fixed schema, pre-defined keys are stored, with foreign key concepts relationships between tables can be easily visualized.

But in MongoDB we have no foreign key concepts, so here we have different client driver approaches to creating Database References. In order to determine relationships between collections using DBRefs we need to scan the documents. To solve all these MongoDB problems, we need to use visualization tool. There are few different Mongo Shell helpers that will give us an idea about general structure of collections about data types, fields name etc. These existing tools have some limitations and some advantages also. By taking these advantages and to overcome disadvantages we proposed a new GUI tool with the help of Mongo Inspector using Python Tkinter.

Some of these tools are explained as follows.

1. Schema.js
2. Variety
3. Mongo Inspector using map Reduce

2.1 Schema.js

This is a schema analysis tool for MongoDB[27] that includes analysis feature in MongoDB shell and providing a new function called schema () with the following signature. This schema function accepts all the same parameters that map reduce function does.

Example:

Create a users collection in database with different schemas of documents.

1) db. users. Insert ({'name': {'first' : 'Ram', 'last' : 'Kumar'}, 'isRegistered': false, 'tags' : ['male']});
2) db.users.insert({'name' : {'first' : 'Mohit', 'last' : 'Kansal'}, 'isRegistered' : false, 'tags' : ['male','new']});
3) db.users.insert({'name' : {'first' : 'Sagar', 'last' : 'garg'}, 'isRegistered' : 1, 'tags' : ['female']});
4) db.users.insert({'name' : 'karan kansal', 'isRegistered' : '0', 'tags' : ['male']});

> // Print our results to the console

Command: db.users.schema();

Processing 4 document(s)...

```
{
  "results" : [
    {
      "_id" : "_id",
      "value" : {
        "wildcard" : false,
        "types" : ["objectid"],
        "results" : [
```

[27] https://github.com/mongodb-js/mongodb-schema

{"type" : "all","docs" : 4,"coverage" : 100,"perDoc" : 1},
{"type" : "objectid","docs" : 4,"coverage" : 100,"perDoc" : 1}
] } },
{
"_id" : "isRegistered",
"value" : {
"wildcard" : false,
"types" : ["boolean","number","string"],
"results" : [
{"type" : "all","docs" : 4,"coverage" : 100,"perDoc" : 1},
{"type" : "boolean","docs" : 2,"coverage" : 50,"perDoc" : 1},
{"type" : "number","docs" : 1,"coverage" : 25,"perDoc" : 1},
{"type" : "string","docs" : 1,"coverage" : 25,"perDoc" : 1}
] } },
{
"_id" : "name",
"value" : {
"wildcard" : false,
"types" : ["bson","string"],
"results" : [
{"type" : "all","docs" : 4,"coverage" : 100,"perDoc" : 1},
{"type" : "bson","docs" : 3,"coverage" : 75,"perDoc" : 1},
{"type" : "string","docs" : 1,"coverage" : 25,"perDoc" : 1}
] } },
{
"_id" : "name.first",
"value" : {
"wildcard" : false,
"types" : ["string"],
"results" : [
{"type" : "all","docs" : 3,"coverage" : 75,"perDoc" : 1},
{"type" : "string","docs" : 3,"coverage" : 75,"perDoc" : 1}
] } },
{
"_id" : "name.last",
"value" : {
"wildcard" : false,
"types" : ["string"],
"results" : [
{"type" : "all","docs" : 3,"coverage" : 75,"perDoc" : 1},

{"type" : "string","docs" : 3,"coverage" : 75,"perDoc" : 1}
] } },
{
"_id" : "tags",
"value" : {
"wildcard" : false,
"types" : ["array"],
"results" : [
{"type" : "all","docs" : 4,"coverage" : 100,"perDoc" : 1},
{"type" : "array","docs" : 4,"coverage" : 100,"perDoc" : 1}
] } },
{
"_id" : "tags.$",
"value" : {
"wildcard" : true,
"types" : ["string"],
"results" : [
{"type" : "all","docs" : 4,"coverage" : 100,"perDoc" : 1.25},
{"type" : "string","docs" : 4,"coverage" : 100,"perDoc" : 1.25}
] } }],
"timeMillis" : 16,
"counts" : {"input" : 4,"emit" : 26,"reduce" : 7,"output" : 7},
"ok" : 1,}

But the limitation of using schema script is that it returns bson not any object and it will also return number long rather that number etc.

2.2 Variety

This is a light weight tool which helps us to get a sense of our application's schema[28]. Here taking an example of user's collection: It provides ease to learn that how data is structured in a collection. It also finds all keys either used as rare or used multiple times[29]. db.users.insert({name: "Tom", bio: "A nice guy.", pets: ["monkey", "fish"], “someWeirdLegacyKey”: "I like dogs!!"});

---

[28] https://github.com/variety/variety
[29] http://blog.mongodb.org/post/21923016898/meet-varietyaschemaanalyzerformongodb

db.users.insert({name: "Dick", bio: "I swordfight.", birthday: new Date("1974/03/14")});

db.users.insert({name: "Harry", pets: "egret", birthday: new Date("1984/03/14")});

db.users.insert({name: "Geneviève", bio: "Çanva"});

db.users.insert({name: "Jim", someBinData: new BinData(2,"1234")});

Command:

$ mongo test --eval "var collection = 'users'" variety.js

Result:

Key	types	occurrences	percents
_id	ObjectId	5	100.0
name	String	5	100.0
bio	String	3	60.0
birthday	String	2	40.0
pets	Array,String	2	40.0
someBinData	BinData-old	1	20.0
someWeirdLegacyKey	String	1	20.0

It looks like "pets" can be either an array or a string.

It solves the limitation of JSON output of schema.js. But inference of relationships isn't supported by both variety and schema.js yet.

2.3 Mongo Inspector using Map Reduce

Mongo-inspector is a library that analysis the data of a MongoDB database to extract its schema. Because MongoDB is a schema less and it has JSON documents and mongo inspector is better tool because it gives output as JSON format[30].

To install mongodb inspector we use following command:

$ Pip install mongo-inspector

[30] https://pypi.python.org/pypi/mongo-inspector/0.2

Usage

```
Import mongo_inspector
Schema = mongo_inspector.extract_schema (
db_name='dbname',
Host= 'localhost'
port='27017')
```

Output

```
{
u'SomeCollection': [
Attribute (name=u'id', types=[u'String']),
Attribute (name=u'someattribute', types=[u'String'])
],
u'AnotherCollection': [
Attribute (name=u'_id', types= [u'ObjectId']),
Attribute (name=u'someattr', types= [u'Object']),
Attribute (name=u'someattr.nested', types= [u'Number']),
Attribute (name=u'somelist', types= [u'Array']),
Attribute (name=u'somelist.__item__', types= [u'Object']),
Attribute (name=u'somelist.__item__.nested',
Types= [u'String', u'Number'])
]
}
```

Attribute (name, types) is just a 'name tuple'. Each attribute can have several types.

It gives an output of schema in the form of tuple. It solves the limitations of both schema.js and variety. It is better tool than above described both tools because it also infer the relationships between collections.

So in this chapter the tools which are already used for visualization of schema is discussed. Here in thesis, by use of MongoDB inspector tool and with the help of python Tkinter a GUI based tool is implemented to extract and analyze the schema of MongoDB database. So in next chapter about research gap between existing and targeted work is going to be discussed. Also Problem statement, that why need to analyze schema of a database of MongoDB is defined.

Chapter 3

Research Questions and Problem Statement

3.1 Gaps between existing and targeted work

So as in above chapter the tools which are already used for visualization of schema is discussed. Schema Analysis or visualization of schema plays an important role for IT professionals who are currently using MongoDB database. As MongoDB gains its popularity very soon and has more number of customers. But it has dynamic, flexible schema. It does not provide any view of structure of data, its field types. A lot of work has already has been done in the visualization of schema. Since people always want GUI tools which are easy to use for their work, also waste less amount of their time. So GUI based schema extractor is important.

After understanding different schema analyzers and with some modifications in the design of existing schema analyzer tool MongoDB Inspector, a GUI based tool has been purposed with the help of python tkinter tool. Results of the analysis are presented and conclusions are drawn on the basis of result, which concludes that the purposed GUI tool performs better that existing command line visualization schema analyzer tools.

3.2 Problem Statement

Objectives of research is to solve the following define problems.

1. Traditionally having limited amount of data but now a days data is increasing at very high speed, having variety of data in a large amount of volume. So, data is cannot be stored in one database. It is very difficult and expensive to collect the data in one data stores. So, integration the data is needed. Developers, DBAs uses Mediator wrapper approach for database integration. For that integration extraction of schema and knowledge about schema structure is needed.
2. DBAs also want to keep track of schema what sort of data is kept in each collection, also want to keep all records of the layout that will help development team plans with ease future expansion.

3. In an organization if someone created the database for an application. Suppose employee who created database has left the company. Now as only he knows what he has created and stored fields in data stores. It means company is totally dependent upon on that employee without having any view of data store schema. So to make organization employees independent from other employees, a view of schema is must, that what is stored in database.
4. All coming from relational databases background, in front of them , non-relational databases seems counter intuitive or odd because some non-relational databases are document, graph, key-value, also have different method of extracting and querying data. Also these data stores are schema less. So to provide better understandability of these data stores at schema level, extraction of schema is needed. It gives a proper view, how keys and values are stored in data stores. It also reduces the gap between relational and non-relational databases. It enables us to define what type of validations applied on data items.

In this chapter research gaps between targeted and existing work is discussed. Then the problems we found while working with MongoDB database is explained, which can be solved easily by analysis of schema or visualization of schema. In the next chapter 4 'Motivation by a case Study', an example of case study of Hospital Management System of MongoDB database, is explained.

Chapter 4

Motivation and Case study

Motivation of creating a GUI tool is to solve the problems defined in previous chapter. To extract schema of a database of MongoDB, a case study of hospital is considered. In this Hospital case study it has many collections to manage different portions of a hospital. ER model of schema of database is explained as follows.

4.1 ER Model

This is a model which is used to describe information of collections of database that how they are relate to each other [20]. This is a graphical way to represent the information of collections.

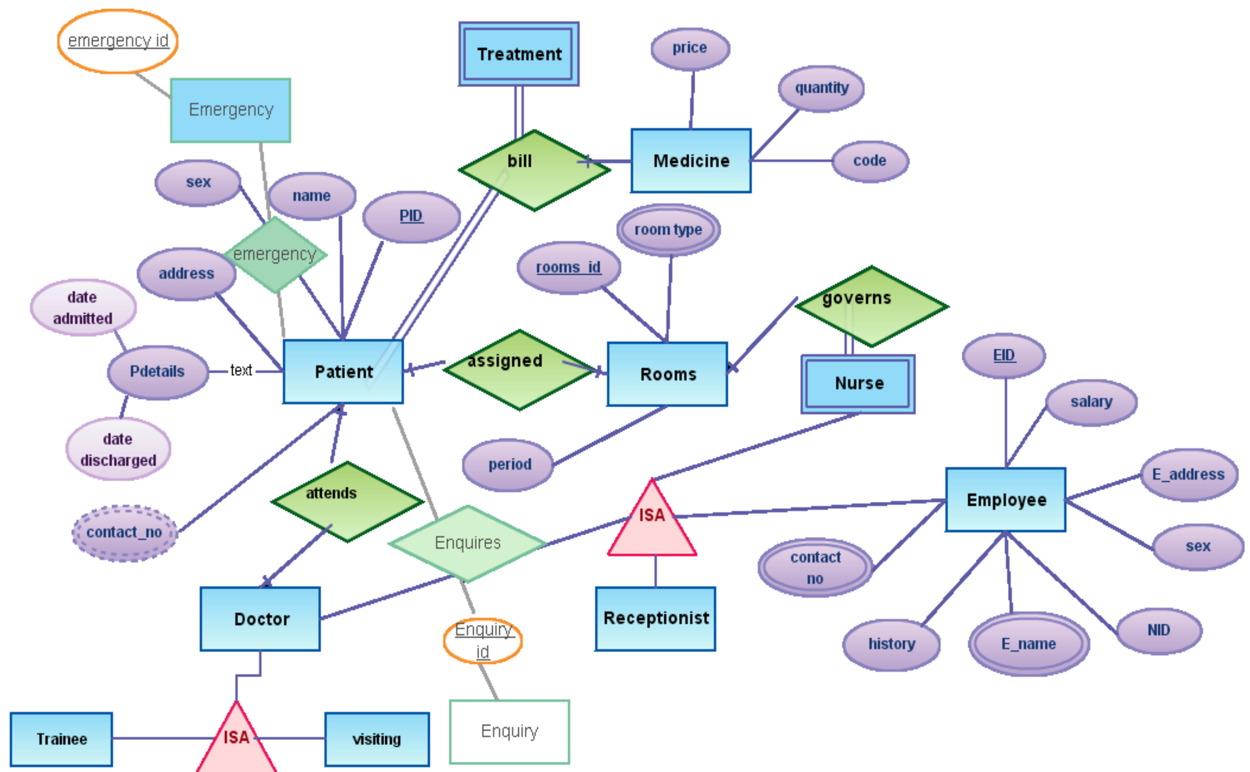


Fig 4.1: ER Diagram of Hospital Case Study

As shown in this Fig Hospital case study contain 10 different collections Patients, Doctors, Nurses, Reception, Cashier, Treatment, Medicine Treatment, Rooms, Radiology, and Pharmacy. These collections are represented as rectangular in this Fig, the ellipses are the fields in each collection, and pointing arrows describe the relation between document to document inside another collection. Each collection has one primary key. The primary key for a collection is represented by underline of the field name.

The collections inside the database are going to be explained in detail. When a patient wants to admitted in hospital, he/she may some time to get enquiry about hospital details so there is an enquiry collection. After that he/she has to register on reception so here is a reception collection. Sometimes there is an emergency of a patient for that emergency collection. There is a patient collection who is admitted in hospital, having some symptoms, and having some personal details in this collection. There is a doctor collection that will treat patients, and give some treatment that information is stored in Medical treatment collection. For test details there is a radiology, Laboratory collections.

Sometime patient has to admit in hospital, for that a room collection, bed collection is needed. To take care of a patient a nurse collection is made. For medicines there is a pharmacy collection. After that when a person is discharged he/she has to pay bills for that cashier collection, it may be happened that he/she is an insured for that there is an insurance collection. There are many persons in hospital who worked as an employee in different departments, to store that information also, for that employee collection is needed and for their salary information, there is a payroll collection.

4.2 Class Diagram

Class diagram is a Unified Modeling Language (UML) in which structure of a system is described, by showing the contents of that system, attributes, and relationships between objects. Class diagram can be used for database description.

Every class diagram contains 3 parts. The upper one is the name of the class, it has to be bold and centered, the middle one is the attribute part that gives the

attributed for that class, it has to be left aligned, and the last part is the methods or operations description. The last part is not needed for database description.

Class diagram could be used for representing the data in MongoDB as shown in Fig 4.2.

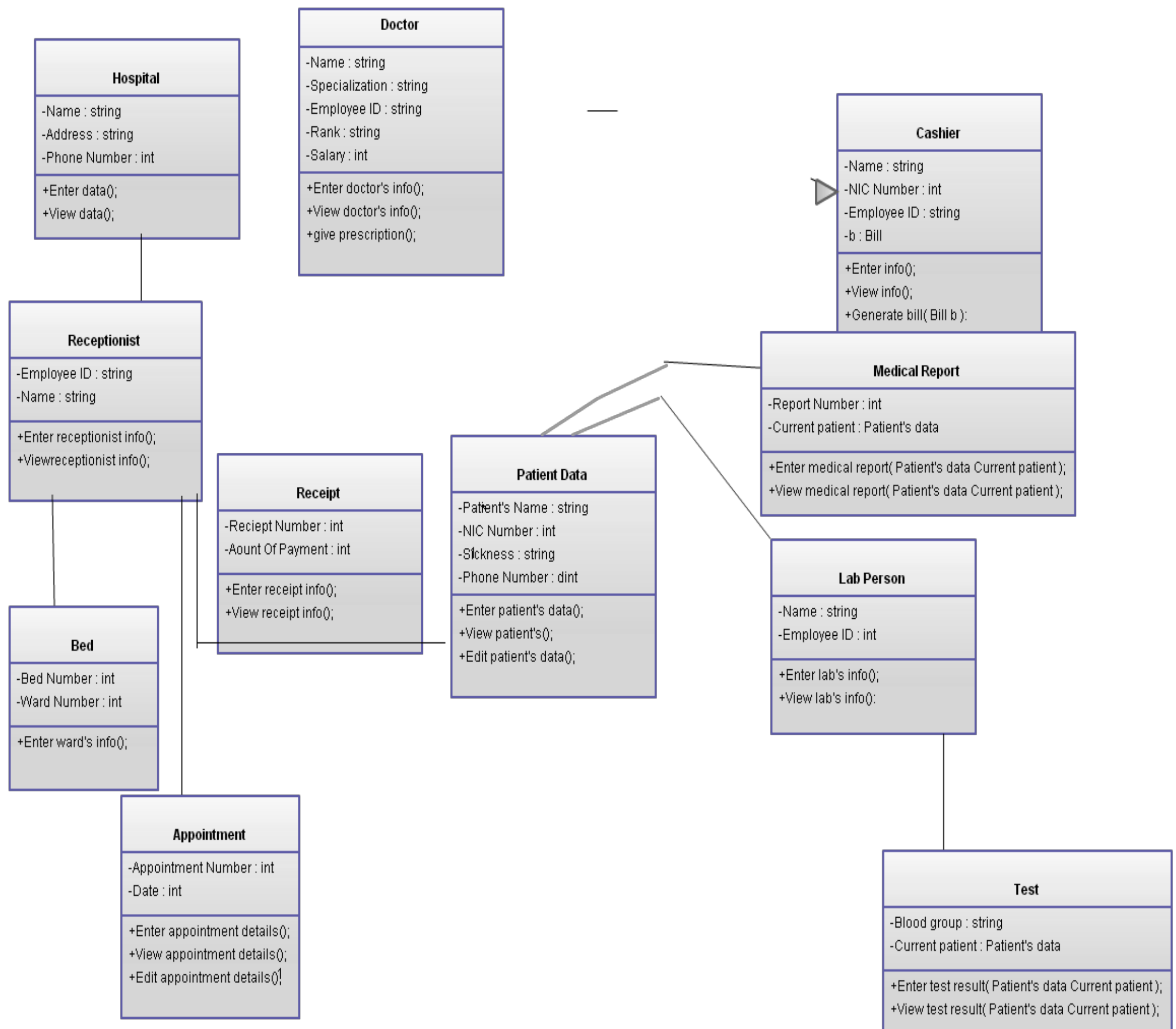


Fig 4.2: Class Diagram of Hospital case study in MongoDB

Collections are represented by classes, the name of the collections is in the upper part of each class, fields names are in second part of the class, and referencing between collections are represented by relationships that connect the classes.

Every collection has primary key field, and this field is represented by (#) before that field name, the fields without any sign before them but having FK after them, they contain the foreign keys from other collections, other fields with (+) sign are the normal fields that contain normal documents of that collection. For the connections between collections are represented by lines, every line has two values one next to the start of it and the second one next to the end of it. If (1) on both sides of the line between two collections, that mean for each document in the first collection an only one document on the second collection, but if (*) on both sides of the line between two collections, that mean one or more document from the first collections could have one or more from the second collection, and if on side (1) and the other side (*) on the lines, that is mean one document from the first collection can has multiple documents from the second collection.

Chapter 5

Implementation and Algorithm

To extract schema we need to do some prior setup, some installations of files. We need also a test database including collections. We also need to install python with tkinter GUI. We also need to get prior knowledge to use tkinter programming language. The basic steps that to be needed to follow to extract schema are given in Section 5.1.

5.1 Steps

Steps to extract schema of database stored in mongodb:

1) Install mongodb

2) make a database hospital and make some collections in that database

```
>Mongo
> use hospital
> db.patient.insert({ })
> db.doctor.insert({})
```

3) Install python

```
> sudo apt-get install idle -python3.4
```

4) To connect python to mongodb database use following commands and use some drivers like pymongo

```
Import pymongo
From bson.objectidimportObjectId
Connection=pymongo.Connection ()
Db=connection ["patient"]
Hospital=db["hospital"]
```

5) As mongodb is schemaless, so to extract schema of mongodb of all collections there is need of mongodb inspector.

6) Install mongodb inspector

7) Now make a GUI with use of python tool tkinter

8) Install tkinter

9) To make a tree view, use of treeview widget in tkinter

10) To make graph view, use of line clipping and rectangle widgets

In Section 5.1 we explained steps to be followed. Logic that follows to implement this concept is explained in Section 5.2.

5.2 Algorithm

Install mongo inspector with the use of command on terminal. then need to import mongo inspector to use this in code. In mongoInspector we have a schema function that accepts some basic parameters and optional parameteres including databse name, host name, port number. Pass these arguments in schema function. This schema function returns a tuple then to convert tuple to dictionary write some code of python programming language.

1. Install mongo inspector by using command “$ pip install mongo-inspector”.
2. Then Import mongoInspector in python code by using command “import mongo_inspector”.
3. Pass MongoDB database and its details to extract schema function of mongoinspector
   Schema = mongo_inspector.extract_schema(db_name='mydb',host='myhost', # optional: default 'localhost' port=xxxx # optional: default 27017).
4. for each in schema:
   1. for k in schema [each]:
      nodeDict[each.encode('utf-8')]. Append (k[0]. Encode ('utf-8'))
5. nodeDict is a dictionary contains field name as a key and its data type as a value.
6. To create tree view of schema of collection use tree widget of python tkinter
   Tree=ttk.Treeview(root1, height=20, columns=3)
   Tree. Heading()
   Tree.column()
   for t in nodeDict:
   tree.insert(column number, row number, values=t)

7. To create graph (relational view) of collection use rectangle and line widget of python tkinter.

```
w=Canvas (root,width=1000,height=600)
w.grid(row=0)
for k in nodeDict[key]:
        r=w.create_rectangle(30,40,40,50)
        t=Label (w,text=k)
```

8. To print all available databases

```
client=MongoClient()
dblist=client.database_names();
print dblist
```

9. To print referred collection

```
for every in myDoc:
        if(every.has_key(list)):
                collectionname= (every[l]).collection
```

10. To print referred database

```
databasename=(every[l]).daabase
```

11. To print all collections

```
def toolbar1():
  for key in (nodeDict.keys()):
        buttton(key)
```

To create tree view we can use tree view widget and for graph view we can use rectangle. To print all available databases cilent.database_names function is used.In this way we can implement our GUI tool.

Chapter 6

Results

In this chapter some snapshots of GUI is shown that has been designed with the help of MongoDB as a database and Python as a web design and development language. First snapshot is a simple design for a home page of the Hospital management.

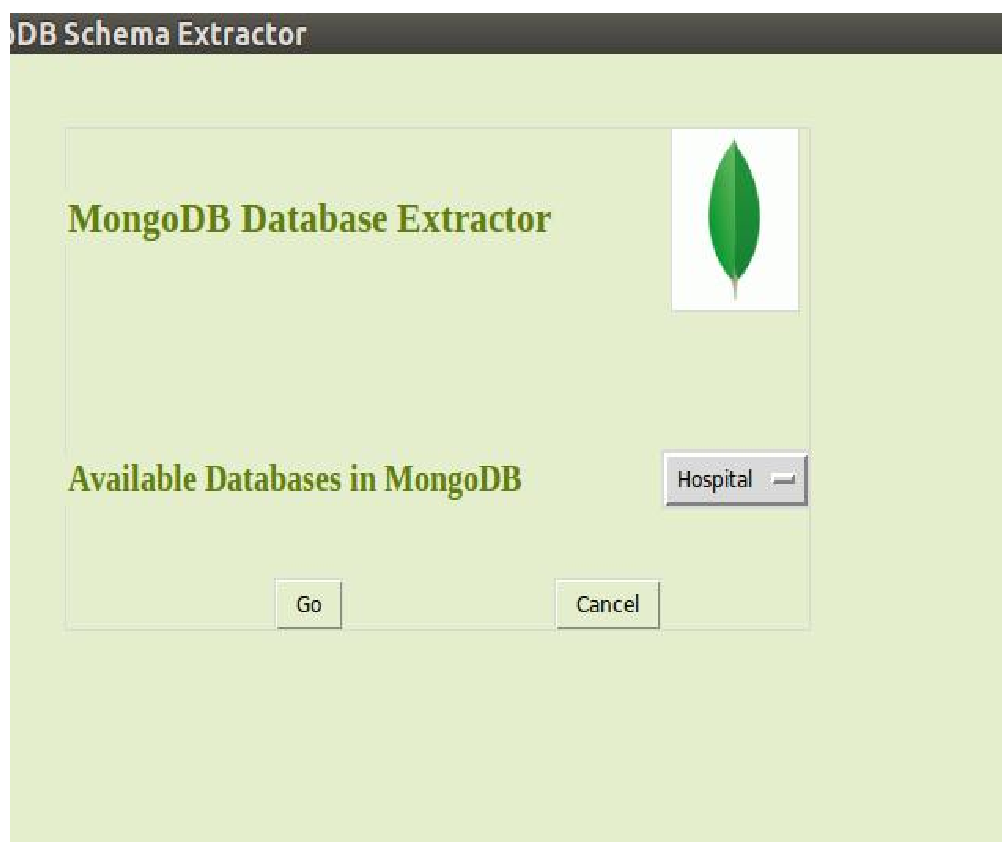


Fig 6.1: Home page of the UI

Fig 6.1 is showing available databases in MongoDB in the form of drop down list. From here choose any available database whose schema wants to be extracted. Otherwise if don't want to extract schema, click on cancel. All collections in selected database is shown in Fig 6.2 in menu bar.

Fig 6.2: Representing all collections of selected hospital database

This snapshot is showing all collections containing in selected database. As here in above snapshot hospital database is chosen. So on this page it is showing related collections stored in hospital database available in MongoDB.

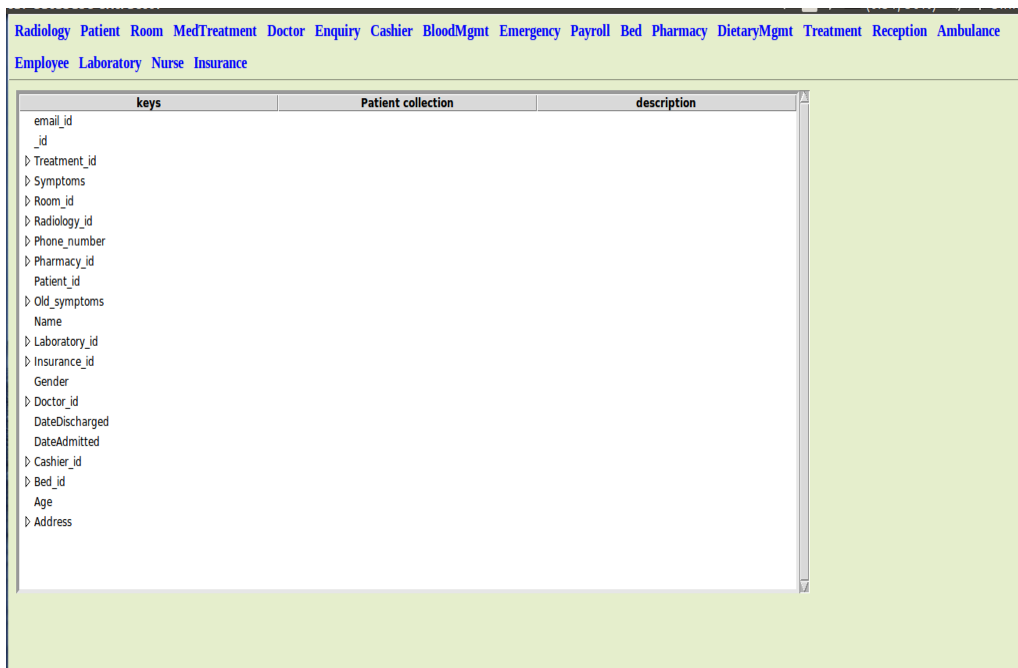


Fig 6.3: Tree View of Patient Collection

Tree view which is representing all keys is shown in Fig 6.3. All the information which is stored in Patient collection in the form of keys is shown in tree view page. It is also representing the nesting fields which are related to another collection. Also it will give the description of the key field whether it is an array item, nested field or embedded field. If it is related to another collection then it will describe that through which id it is related to which collection and in which database. It describes the whole structure.

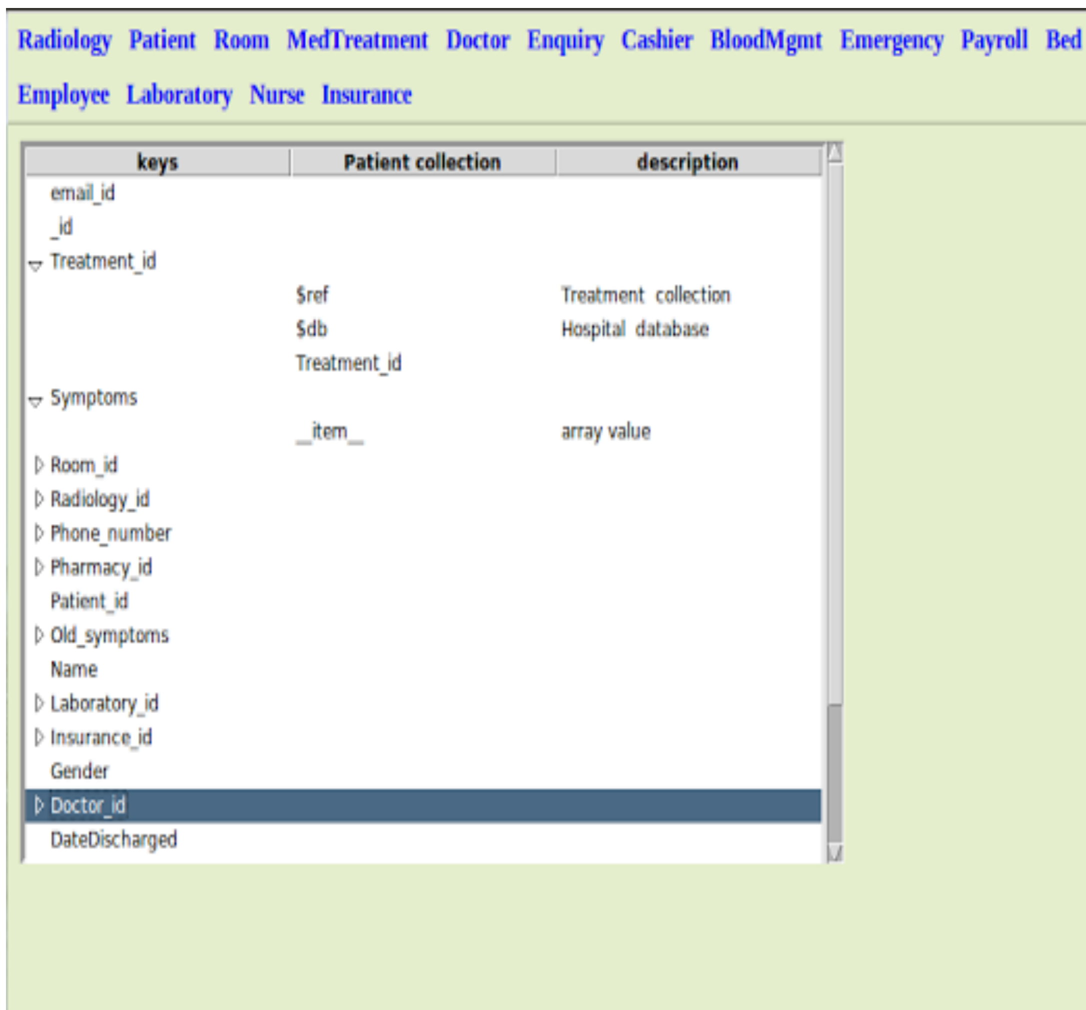


Fig 6.4: Detailed Tree View of Patient Collection

This snapshot shows tree view with full description; it gives the view that which is nested document, containing which fields. Which field is an array as it gives _item_ name to array.

Relational view or Graph view is shown as in Fig 6.5.

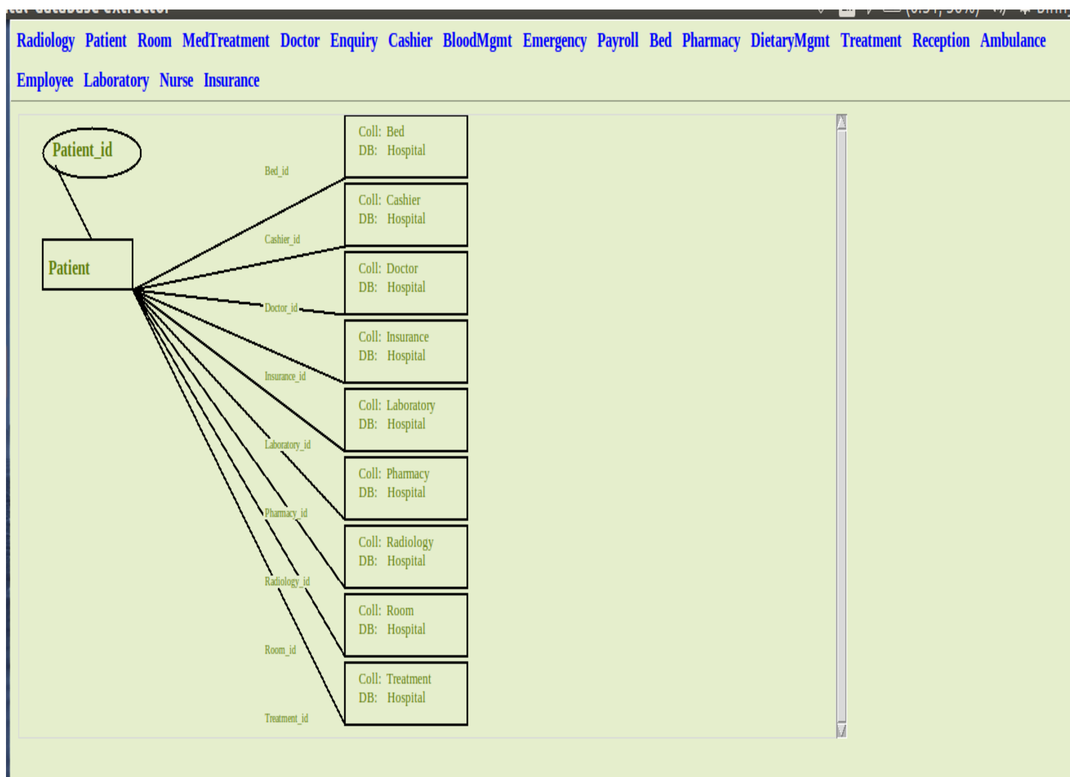


Fig6.5: Graph view of Patients information page

This snapshot shows all information that is which collection is related to which collection graphically. As in this snapshot patient collection has key field patient_id which is used as primary key and it is showing that to which collections they are connected.

Chapter 7

Conclusions and Future Scope

7.1 Conclusions

NoSQL databases have many advantages over relational databases so more and more people are opting for this. Also, Relational databases are not best fit for unstructured data storage whereas NoSQL databases are more efficient, since they store data in various formats. Here in this thesis, much focused on schema of document oriented database, which is very appropriate for web applications, which can store unstructured data and handle dynamic queries.

MongoDB is a popular database under document oriented databases and provides horizontal scalability, high performance, flexibility and security. As data is spread across different sites so we need to integrate the data from different sites. But proper knowledge is required to integrate data at schema level. In this thesis, Extraction of database schema using MongoDB as backend and Python tkinter for frontend with the use of Mongo Inspector is implemented. It gives internal structure of a collection. With this DBAs can easily managed their databases. It gives tree view, table view and graph view of a collection of database.

In the end important features of GUI extraction of schema tool is described as it is a GUI based tool and easy to use. We can understand internal schema what keys and what types of that keys are stored. It can be easily manageable. It defines the structure properly. It gives JSON view schema about which collection relates to other collection. This tool will be very helpful for DBAs to manage their document oriented databases.

7.2 Future Scope

In this thesis, only extraction of schema of a database of MongoDB, this is sub part of need of data integration. The field of extraction of schema GUI can be further explored in the light of following suggestions .We can make GUI better than this in future. We can implement relational view between all collections which should be equivalent to ER diagrams of relational database. We will explore more Data integration concept.

Chapter 8

Conclusion and Future Scope

A possible approach to extract schema from a graph datastore using a sample EVD database has been presented and implemented successfully, also providing a way to process the extracted schema in a universal manner using datalog. Various tools that were used for converting graph data into GraphML snapshot, then extracting and transporting the schema to datalog were also shown at glance. With the help of these tools the whole process is simplified and in the end a valid query is formulated after consulting fact base. Our approach equips a NoSQL datastore specifically graph datastore to provide schema but is not limited to single datastore only. As the methodology proposed tends to use and deploy only universally supported measures most of the other graph database service providers also comply with it. GraphML and Datalog being supported by all boosts the applicability span of the approach.

We intend to extend our work to include other graph datastores initially, then exploring other datastore classes and storage models as well using the same methodology. As of now the whole process works on a passive instance of the datastore, exploiting merely a snapshot of the real data. This can restrict the functioning if some of the data in the instance is manipulated, which means on every major change in graph database the schema extraction process will have to be repeated in current methodology. So in order to tackle with this we will develop a way to interact with the live data and maintaining the schema in real-time, instead of snapshot. Systematic improvements and extensions to the proposed work could go a long way towards integrating heterogeneous data models, combining their potentials, and polyglot persistence.

Bibliography

[1] Luke Welling and Laura Thomson. *PHP and MySQL Web development*. Sams Publishing, 2003.

[2] Kevin Loney. *Oracle Database 11g The Complete Reference*. McGraw-Hill, Inc., 2008.

[3] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[4] The couchdb website, 2013. URL `http://couchdb.apache.org/`.

[5] Jeremy Zawodny. Redis: Lightweight key/value store that goes the extra mile. *Linux Magazine*, August 2009.

[6] Tim O'reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

[7] Karamjit Kaur and Rinkle Rani. Modeling and querying data in nosql databases. In *Big Data, 2013 IEEE International Conference on*, pages 1–7. IEEE, 2013.

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[10] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[11] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[12] Renzo Angles. A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 171–177. IEEE, 2012.

[13] Daniel J Abadi, Peter A Boncz, and Stavros Harizopoulos. Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665, 2009.

[14] Hbase homepage, 2013. URL `http://hbase.apache.org/`.

[15] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[16] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008.

[17] Jans Aasman. Allegro graph: Rdf triple database. Technical report, Technical report. Franz Incorporated, 2006. ur l: http://www. franz. com/agraph/allegrograph/(visited on 10/14/2013)(cited on pp. 52, 54), 2006.

[18] Flockdb homepage, 2014. URL `https://github.com/twitter/flockdb`.

[19] Claudio Tesoriero. *Getting Started with OrientDB*. Packt Publishing Ltd, 2013.

[20] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.

[21] Rainer Manthey. Datalog and beyond: A" gentle" introduction to research in deductive databases.

[22] Krzysztof J Cios and G William Moore. Uniqueness of medical data mining. *Artificial intelligence in medicine*, 26(1):1–24, 2002.

[23] Tracy D Gunter and Nicolas P Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of Medical Internet Research*, 7(1), 2005.

[24] Ken Ka-Yin Lee, Wai-Choi Tang, and Kup-Sze Choi. Alternatives to relational database: comparison of nosql and xml approaches for clinical data storage. *Computer methods and programs in biomedicine*, 110(1):99–109, 2013.

[25] Marko A. Rodriguez and Peter Neubauer. The graph traversal pattern. *CoRR*, abs/1004.1001, 2010.

[26] Trevor M. O'Brien, Anna M. Ritz, Benjamin J. Raphael, and David H. Laidlaw. Gremlin: An interactive visualization model for analyzing genomic rearrangements. *IEEE Trans. Vis. Comput. Graph.*, 16(6):918–926, 2010. URL `http://dblp.uni-trier.de/db/journals/tvcg/tvcg16.html`.

[27] Peter T. Wood. Query languages for graph databases. *SIGMOD Record*, 41 (1):50–60, 2012.

[28] Eman Yasser Daraghmi and Yuan Shyan Ming. Using graph theory to re-verify the small world theory in an online social network word. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, pages 407–410. ACM, 2012.

[29] Florian Holzschuher and René Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204. ACM, 2013.

[30] Marie-Christine Rousset and Chantal Reynaud. Knowledge representation for information integration. *Information Systems*, 29(1):3–22, 2004.

[31] Ramez Elmasri. *Fundamentals of database systems*, volume 2. Pearson Education India, 2007.

[32] Peter Mc Brien and Alexandra Poulovassilis. A formal framework for er schema transformation. In *Conceptual Modeling—ER'97*, pages 408–421. Springer, 1997.

[33] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods—a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007.

[34] Bo Fu, Natalya F Noy, and Margaret-Anne Storey. Indented tree or graph? a usability study of ontology visualization techniques in the context of class mapping evaluation. In *The Semantic Web–ISWC 2013*, pages 117–134. Springer, 2013.

[35] Hondjack Dehainsala, Guy Pierra, and Ladjel Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Advances in Databases: Concepts, Systems and Applications*, pages 497–508. Springer, 2007.

[36] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *SemWeb*, 2001.

[37] Yassine Drias. Towards a model of storage oriented nosql for an ontology database. 2014.

[38] Jianhou Gan, Bin Wen, and Jinxu Li. Research on ontology mapping of tourism information resources based on description logic.

[39] Marco Alfonse, Mostafa M Aref, and Abdel-Badeeh M Salem. An ontology-based system for cancer diseases knowledge management. 2014.

[40] Alejandra González-Beltrán, Ben Tagger, and Anthony Finkelstein. Ontology-based queries over cancer data. *arXiv preprint arXiv:1012.5506*, 2010.

[41] N Noy and Deborah L McGuinness. Ontology development 101. *Knowledge Systems Laboratory, Stanford University*, 2001.

[42] Natalya F Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W Fergerson, and Mark A Musen. Creating semantic web contents with protege-2000. *IEEE intelligent systems*, 16(2):60–71, 2001.

[43] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[44] Xiaolei Qian. Semantic interoperation via intelligent mediation. In *Research Issues in Data Engineering, 1993: Interoperability in Multidatabase Systems,*

1993. Proceedings RIDE-IMS'93., Third International Workshop on, pages 228–231. IEEE, 1993.

[45] William Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[46] Jeffry D Ullman. The database approach to knowledge representation. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 1346–1349, 1996.

[47] Fernando Sáenz-Pérez. Des: a deductive database system. *Electronic notes in theoretical computer science*, 271:63–78, 2011.

[48] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[49] Francesca Bugiotti, Damian Bursztyn, Alin Deutsch, Ioana Ileana, and Ioana Manolescu. Invisible glue: Scalable self-tuning multi-stores. In *Conference on Innovative Data Systems Research (CIDR)*.

[50] Yigal Arens, Chin Y Chee, Chun-Nan Hsu, and Craig A Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(02):127–158, 1993.

[51] A Buccella, A Cechich, and NR Brisaboa. Ontology-based data integration methods: A framework for comparison. paper proposal. university of comahue. sources. *Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 2005.

[52] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[53] Raghu Ramakrishnan and Jeffrey D Ullman. A survey of deductive database systems. *The journal of logic programming*, 23(2):125–149, 1995.

[54] Mark Lutz. *Programming python*, volume 8. O'Reilly, 1996.

List of Publications

Manpreet Singh and Karamjit Kaur. SQL2Neo: Moving Health-care Data From Relational to Graph Databases. In *2015 IEEE International Advance Computing Conference* (in proceeding). IEEE, Bangalore, June 12-13, 2015.

YouTube Video Link

https://www.youtube.com/watch?v=2uHRbomkLbM

Reflective Diary

January 2015

In the previous semesters I studied the difference between NoSQL and relational model. Under the guidance of my mentor I had picked NoSQL as my topic for research. I studied in detail all the four classes of NoSQL, where some of them were already explored by me in previous semesters. In the beginning I started with Key-value store based datastore named Redis but Redis being under development and lesser prior exploration by others I faced a great difficulty in finding something to exploit in it. So I had to give up on this topic and decided to look for interest in some other class of NoSQL. Finally with the help of my mentor and after going through many papers I came across Graph datastores and found them interesting and suitable enough to consider it for further exploitation.

Outcome: Chose a topic of my own interest by considering and even failing upon other considerations.

February 2015

Once I was sure about proceeding in Graph datastores, I starting studying various graph datastore implementations. In order to reach to a decision I consulted db-engines.com for viewing the rankings of the various implementations. Most of the papers that I read and the ranking website too rated Neo4j as the most recommended graph datastore implementation. Moreover, the most impressive thing about the datastore was its query language. At first it surely seemed alien to me but as I dived deeper into it, I found that it was pretty much similar to RDBMS's SQL. This month was dedicated learning cypher query language of Neo4j, its difference from SQL, how it can be mapped from or to SQL, transformation or moving of a working database in relational model to graph database. It was during this period only that I made notes for all the new languages and other things I used to learn, that became handy when I started writing paper to be published in conference later.

Outcome: Learnt a completely new language (cypher) and its link with SQL which introduced me to "transportation" topic which partially came into use in the paper I published.

March 2015

While I was searching for a problem statement to work on my mentor introduced me to some topics highly in debate and still unresolved: data integration, polyglot persistence, knowledge-base, etc. I started by studying these topics and came across many new terms and topics that I made note of and studied also. During this time I started looking into various knowledge representation techniques as data integration solely relied on some form of ubiquitous representation for heterogeneous datastores. It was that the base for my current problem statement started taking its shape although I was unknown to it. After studying lot of papers on various knowledge representation techniques I was sure that data integration could be achieved through managing schema from all datastores at a central level with a help of a highly capable "manager" at the core. For this ontologies seemed a great option and I delved into it. I learnt the working of ontologies, the concept of integration being implemented with their help in smart cities, tourism, etc. I started by leaning from basics on how to develop an ontology, their specifications and other details. I also learnt working on a tool named Protegé, used for designing high level ontologies. Thanks to the highly informative video tutorials by Dr. Noureddin Sadawi on Protegé I was able to grasp it really quick.

Outcome: Ontologies are very resourceful in storing and representing data and can also be modeled to integrate and manage schema of heterogeneous datastores.

April 2015

Ontologies were helpful and showed a great promise in the intended schema management work but due to the lack of the present work in the field and due to the missing link between ontologies and NoSQL I had to give up on the topic. But thankfully the last topic left in-between, transportation and transformation

of Neo4j came back into the scene and I decided to apply the schema extraction idea on Neo4j. But in this also I needed a universal approach and in order to do so, with my mentor's help I explored decades old technology of Prolog which has not been used for a long time due to its limited usage in present day scenario. I started learning logical programming, prepositional logic, FOL, Prolog, etc. as they could be used substantially in the role of "manager" of central schema reservoir. I came across an old technology LOOM, read a lot about it but could not get it to work for my approach. After trying my luck with LOOM, LIM, etc. like languages I came across Datalog which is a highly powerful and efficient subset of Prolog. Dr. Christoph Lofi's video tutorials provided a great insight into the working knowledge of Logics and Datalog eventually. With the idea in mind and a process to propose I started writing the paper for conference and with the help of notes I made I finished it and communicated it which got selected for publication later. Then I started looking into the implementation process as the idea had to be materialized as well. Fortunately, when I started working in Python, all the APIs pertaining to Neo4j were available for it and also pyDatalog, a variant of Datalog for Python was also available. With this I began the implementation of the proposed methodology in the paper.

Outcome: Although I stumbled upon a major setback due to non-applicability of much studied ontologies, their knowledge really helped me a lot in the further process and I was able to come up with a problem statement as well as a viable solution for the same.

May 2015

During the implementation phase I faced many issues such as non-conformance of some of the APIs, difficulty in finding a universal way to extract data from Neo4j, difference in the syntax of Datalog and pyDatalog, etc. After a lot of input and constant googling, stackoverlow help and above all my mentor's guidance the system was successfully implemented pending its validation against a live use case scenario. For this I picked medical field scenario and studied in detail Ebola Virus Disease (EVD), it took me a lot of time to create and manage a

live working graph database for the EVD to test the working of my system. Finally the database was ready, returning efficiently the generic simple and complex queries, and I tested the system to generate path for a given query after consulting the schema extracted by the system but the system failed drastically. Recursions in pyDatalog not being debug-able always caused an issue, after a lot of improvisation and inputting great number of days and hours I got the system to work properly.

Outcome: Deployed live working EVD database in Neo4j and developed the system in python to simulate and validate the proposed methodology.

June 2015

By this month I knew that my paper has been select and with few minor changes in IACC conference. Now the only thing left was to write thesis, which I started in this month. I started by browsing through all the papers I had read and created a time-line of the order in which I read them and in the same fasion started writing the second chapter of my thesis "literature survey". It took quite some time and after its completion started working on other chapters, documenting various tools and techniques used, all the way to explaining the implementation with the help of the running EVD example.

Outcome: Successful recognition of proposed work at international conference and completion of thesis work.