Schema Extraction of Document Database - MongoDB

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

Master of Engineering

in

Computer Science and Engineering

Submitted By

Binny Garg

(Roll No. 801332006)

Under the supervision of:

Mrs. 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, "*Schema Extraction of Document Database-MongoDB*", 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 Mrs. Karamjit Kaur and refers other researcher's work which are duly listed in the reference section.

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

Signature:
Binny Garg
Binny garg

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

Karamjit

(Mrs. Karamjit Kaur)

Lecturer, CSED

Countersigned by

Deepak Garg

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

(Dr. S. S. Bhatia)

Dean (Academic Affairs)

Thapar University

Patiala

Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success. First of all I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds. With the profound sense of gratitude and heartiest regard, I express my sincere feelings of indebtedness to my guide **Karamjit Kaur,** Lecturer, Computer Science and Engineering Department, Thapar University for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable co-operation, generous attitude and above all her blessings. She has been a source of inspiration for me. I am grateful to **Dr. Deepak Garg**, Head of Department and **Dr. Damandeep Kaur,** P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and inspiration for the completion of this thesis. I will be failing in my duty if I do not express my gratitude to **Dr. S. S. Bhatia,** Senior Professor and Dean of Academics Affairs in the University, for making provisions of infrastructure such as library facilities, computer labs equipped with internet facility, immensely useful for the learners to equip themselves with latest in the field.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted co-operation helped me in doing this thesis.

Binny Garg

Binny Garg

(801332006)

Abstract

Moving to big data world, where data is increasing in unstructured fashion with high velocity. To store this bundle amount of data there is a need of data-stores. Traditionally relational databases are used, which are now not compatible to handle this large amount of data, so need to move on with non-relational data-stores. But the problem is only one non-relational data store cannot fulfill all needs, so there is a need of more than one non-relational data store according to applications. To use more than one data-store there is a need to integrate data at schema level. As non-relational data stores are schema less so proper knowledge of its schema is required.

The purpose of this thesis is to explain schema extraction of document oriented database MongoDB. Databases including both relational and non-relational are discussed. Then, a brief overview of various existing command line tools used for analyzing and visualizing the schema is provided. The main focus of this study is to create GUI tool with the help of mongo Inspector. Explanation of basic steps must be followed to implement this idea of creating a GUI is also given and description of used Algorithm is provided.

On the basis of the results of this research, it can be concluded that this tool is very helpful for DBA's to manage document databases. This GUI based tool implemented in thesis defines the internal structure of schema in detail as it provides tree view, table view, and graph view of schema which is primarily need of data integration at schema level.

Table of Contents

Certificate ..... **i**
Abstract ..... **ii**
Acknowledgement ..... **iii**
Table of contents ..... **iv**
Introduction **1**
1.1 Evolution of Databases 1
1.1.1 Relational Database Management System (RDBMS) 2
1.1.2 NoSQL 4
1.2 NoSQL Drivers 6
1.3 Data Model Representation 8
1.3.1 RDBMS 8
1.3.2 Document Oriented Data-Store 9
1.3.3 Key Value based Data Store 10
1.3.4 Column Oriented Data Store 11
1.3.5 Graph based Data Store 12
1.4 MongoDB 13
1.4.1 Background and Properties 13
1.4.2 Data Types Used in MongoDB 14
1.4.3 MongoDB Queries 14
1.4.4 Map Reduce in MongoDB 19
1.5 Challenges 21
1.6 Organization of thesis 22
A Brief Review of Existing Schema Extractors **23**
2.1 Schema.js 24
2.2 Variety 26
2.3 Mongo Inspector using Map Reduce 27
Research Questions and Problem Statement **29**
3.1 Gaps between existing and targeted work 29
3.2 Problem Statement 29
Motivation and Case study **31**
4.1 ER Model 31
4.2 Class Diagram 32
Implementation and Algorithm **35**
5.1 Steps 35

5.2 Algorithm 36
Results 38
Conclusions and Future Scope 42
7.1 Conclusions 42
7.2 Future Scope 43
References 45
List of Research Publications 49

List of Figures

Fig 1.1: Conceptual level 3
Fig1.2: External level 3
Fig 1.3: CAP Theorem 5
Fig 1.4 Acid vs Base 6
Fig 1.5: RDBMS 9
Fig 1.6: Document oriented data store 10
Fig 1.7: Key Value based Data Stores 11
Fig 1.8: Column Oriented Data Store 12
Fig 1.9: Graph based Data Store 13
Fig 1.10: Insertion of data 15
Fig 1.11: Searching in MongoDB 16
Fig 1.12: Updation in MongoDB 17
Fig 1.13: Group by and Order by in MongoDB 18
Fig 1.14: Deletion in MongoDB 19
Fig 1.15: Map reduce in MongoDB 21
Fig 4.1: ER Diagram of Hospital Case Study 31
Fig 4.2: Class Diagram of Hospital case study in MongoDB 33
Fig 6.1: Home page of the UI 38
Fig 6.2: Representing all collections of selected hospital database 39
Fig 6.3: Tree View of Patient Collection 39
Fig 6.4: Detailed Tree View of Patient Collection 40
Fig6.5: Graph view of Patients information page 41

Chapter 1

Introduction

"Data", a small word entity, is a collection of raw facts or Figs that are used for references and analysis, while information is a processed form of data. Data itself has no meaning so need to be converted data into valuable information. Example: data as a raw fact is 60, but it does not have any meaning until we say it as weight is 60, now it has some meaning. So data to information transformation is required. There are many models with which data can be converted to information. Database is software where related data can store logically in organized manner.

Also moving to big data world, where data is increasing with high speed, need some storage space to store the data in some formats, so that it can be used properly. Hence database is that storage space. Database management System is software which is used to process information and manage operations like insertion, deletion and updation.

This chapter gives an introduction about databases. Section 1 gives definition of databases, data, and evolution of databases. Whereas Section 2 explains what are the factors by which non-relational databases are so much popular over relational databases. Section 3 briefs about different data representation models. Section 4 gives an idea about the challenges and problems found when using non-relational databases. Section 5 gives an overview of how the thesis is organized.

It has crossed 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. Now moving up with use of non-relational databases. In Section 1.1 evolutions of databases how we moved to non-relational databases from traditional databases is discussed.

1.1 Evolution of Databases

To manage databases, there are many data management paradigms. First one is relational database. But no object oriented concept is found in relational database which is a very big problem. As professionals are using programming languages at

the front end like Java, C++ but on connecting to back end database, mapping of fields described in programming language to database fields is needed. This mapping is known as Impendence Mismatch Problem[1] where classes are mapped to tables, Objects are mapped to rows and attributes are mapped to columns in RDBMS.

In Java/C++ (object oriented languages), objects are created early at the starting of program and destroyed immediately at the end of program. So, this mapping is very difficult. So now it's a time to store objects directly in hard disk and retrieve from them. Storing objects directly is popular as OODBMS[2]. Now it becomes the competitor of RDBMS. But at that time RDBMS is very popular so Oracle Corporation introduced the object oriented features in RDBMS[3]. ORDBMS a new version of RDBMS is introduced.

Now a days data is growing with high speed and data is spreading at different sites geographically. So to handle this data shifting to distributed database systems is needed. Thinking about distributed systems, joining of data from multiple sites is a big problem. An idea came into mind that perform aggregation of data and keep it on one cluster. By keeping data on one cluster, joining operation and normalization is very easy. This idea is popular as NoSQL [1]. Google, automated tracking systems, Facebook they all are using NoSQL to handle their data. A new challenge has come into picture to handle big data for analysis, storage and historical data which becomes the biggest competitor of RDBMS also. Now two popular relational and non-relational databases in detail are going to be discussed.

1.1.1 Relational Database Management System (RDBMS)

RDBMS is a database having tables and rows, was developed by E.F. Codd [2]. The problems of file based approach systems is overcome by this and solves the problems of data dependencies. It has 3 layer architecture, first layer is internal layer that handles the compression, encryption, and decryption and describes the storage scheme of table. Second layer is conceptual layer shown in Fig 1.1 in which schema of the tables, column attributes defined in the table are described. Third layer is the external layer shown in Fig 1.2 which explains views about which part of data is

[1] http://www.service-architecture.com/articles/object-oriented databases/impedance_mismatch.html
[2] http://www.techopedia.com/definition/8715/object-relational-database-management-system-ordbms
[3] https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

shown to which user. There is a primary key to handle the uniqueness of data in a table.

Example: We have a doctor table in which id, doctor_name, specialization, hospital_name are fields and we want to show only id and doctor_name to user. So internal level describes the storage scheme and conceptual level contains its field name and external level describes the id and doctor name as view.

Id	Doctor_name	Specialization	Hospital_name
1	Dr. Anand	Urologist	RG stone
2	Dr. Aulakh	Urologist	DMC Ldh.

Fig 1.1: Conceptual level

Id	Doctor_name
1	Dr. Anand
2	Dr. Aulakh

Fig1.2: External level

ACID Properties

RDBMS is popular due to ACID properties [3] which are very helpful while making transactional or financial applications like in banking, in reservation systems *etc*.

ACID stands for **Atomicity, Consistency, Isolation and Durability**. These four things are the basic requirement in transactional applications.

- ***Atomicity*** means that if any transaction is performing some tasks, that tasks should be performed either 100% or 0% means all or none.
- ***Consistency*** means data should be consistent before the transaction and after the transaction *i.e.* sum of money always should remain same.
- ***Isolation*** is to run multiple transactions at a time in a consistent manner to increase the throughput.
- ***Durability*** means data backup should be there means while performing the transactions if system is crashed then we can redo or undo. This can be achieved by log records and handled by recovery management systems [4].

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.

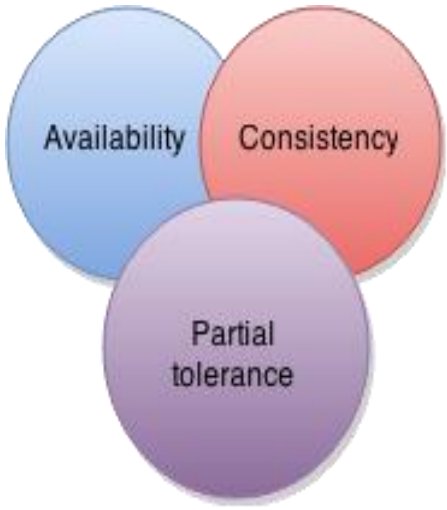

Fig 1.3: CAP Theorem

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/

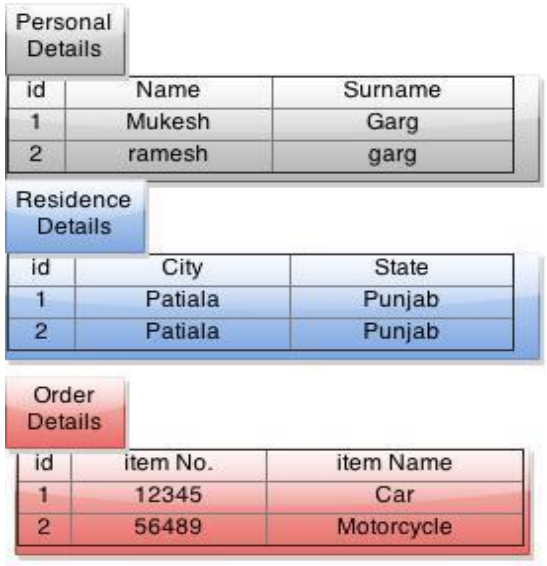

Fig 1.5: RDBMS

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

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

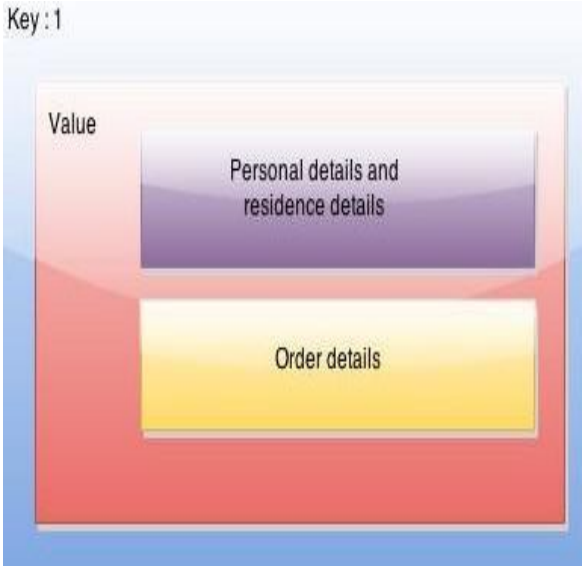

Fig 1.7: Key Value based Data Stores

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

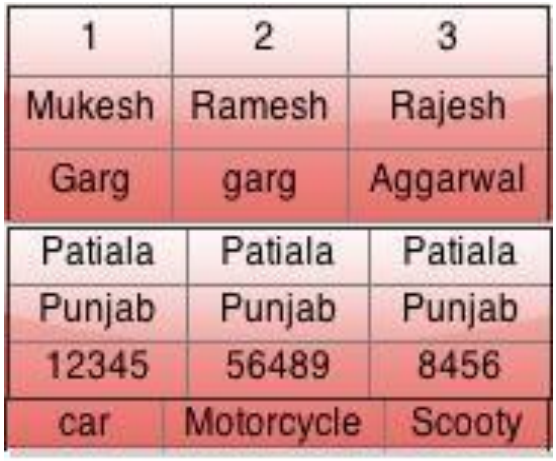

Fig 1.8: Column Oriented Data Store

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/

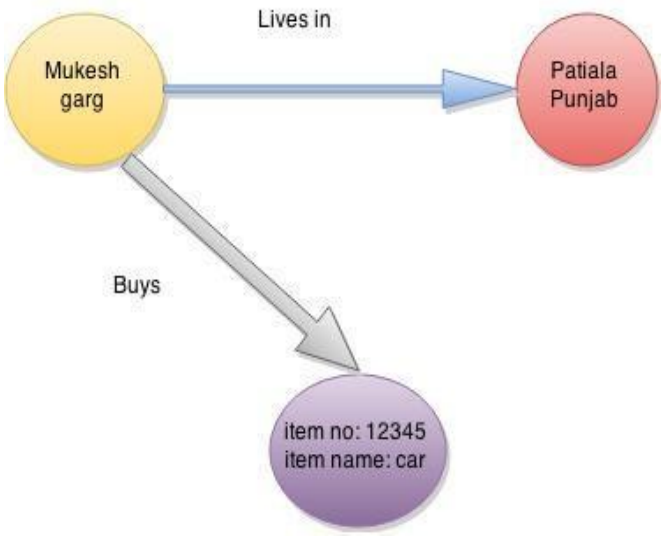

Fig 1.9: Graph based Data Store

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

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.

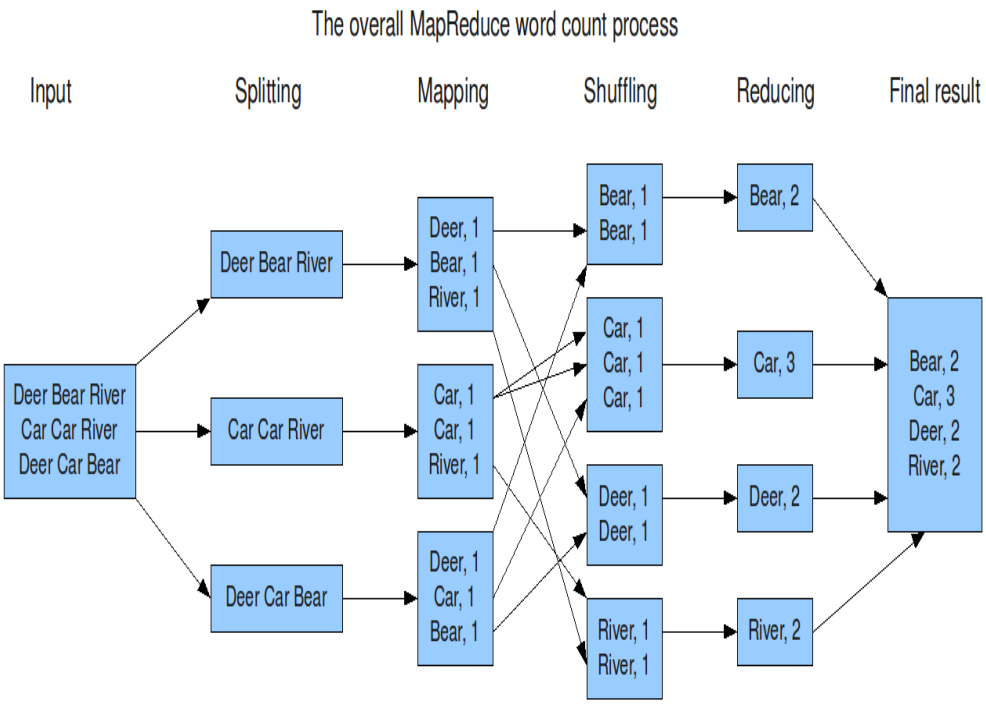

Fig 1.15: Map reduce in MongoDB

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.

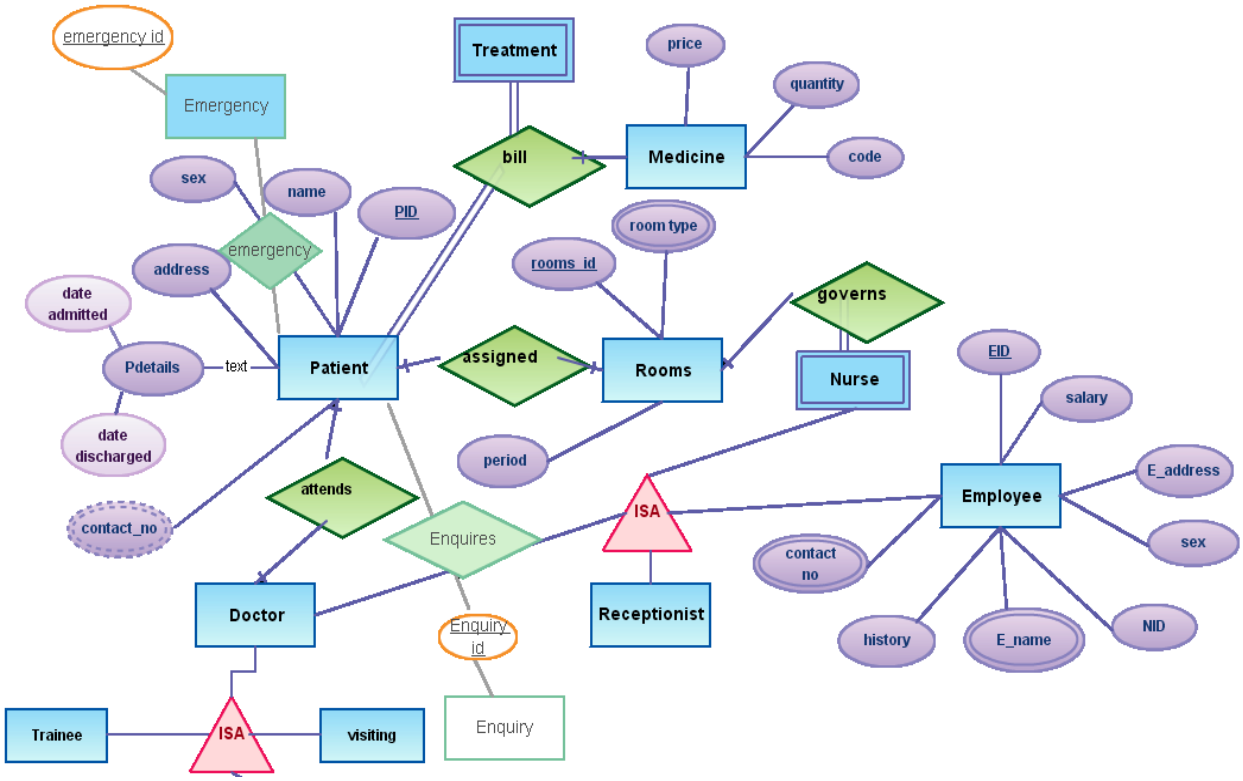

Fig 4.1: ER Diagram of Hospital Case Study

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.

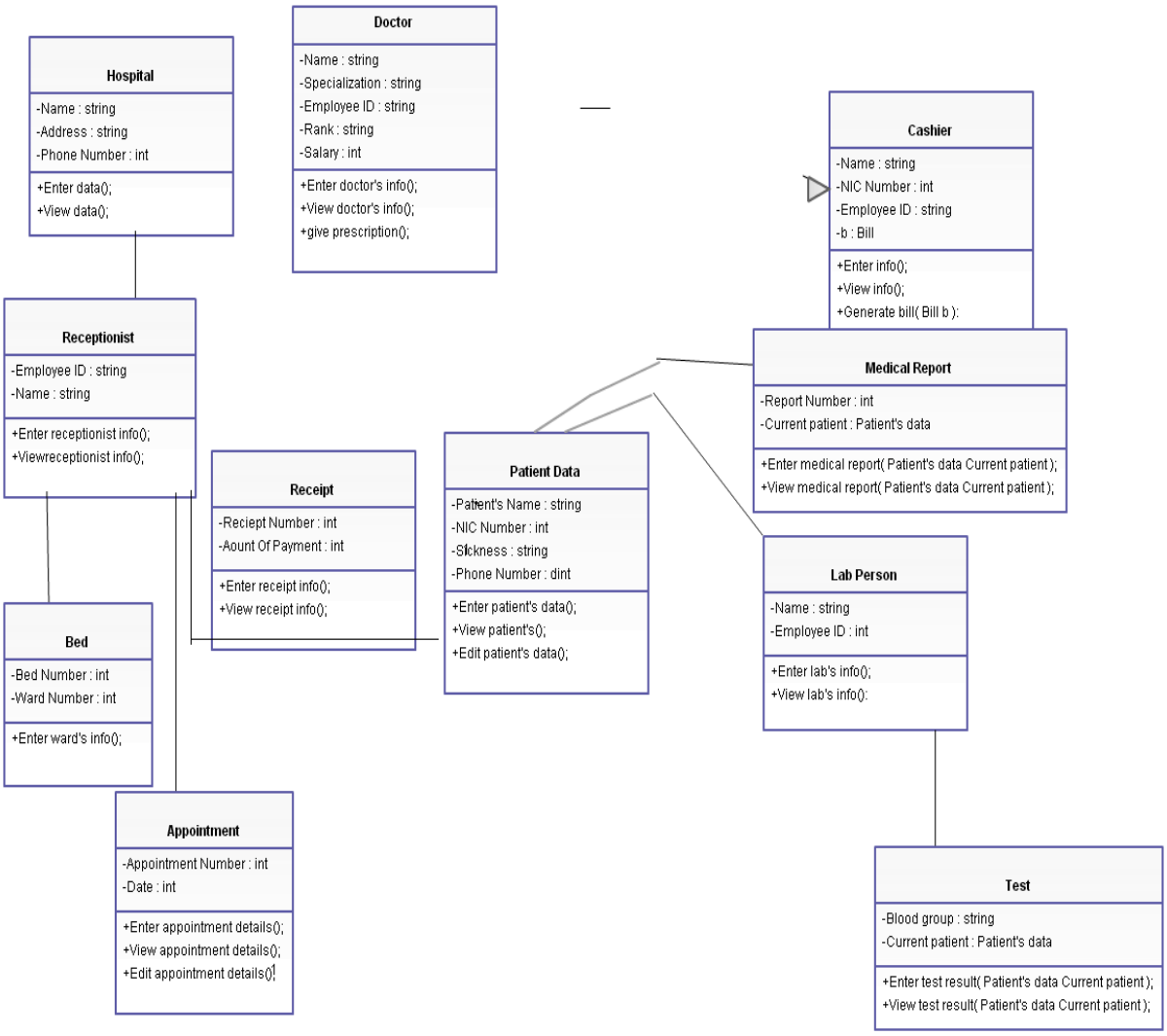

Fig 4.2: Class Diagram of Hospital case study in MongoDB

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.

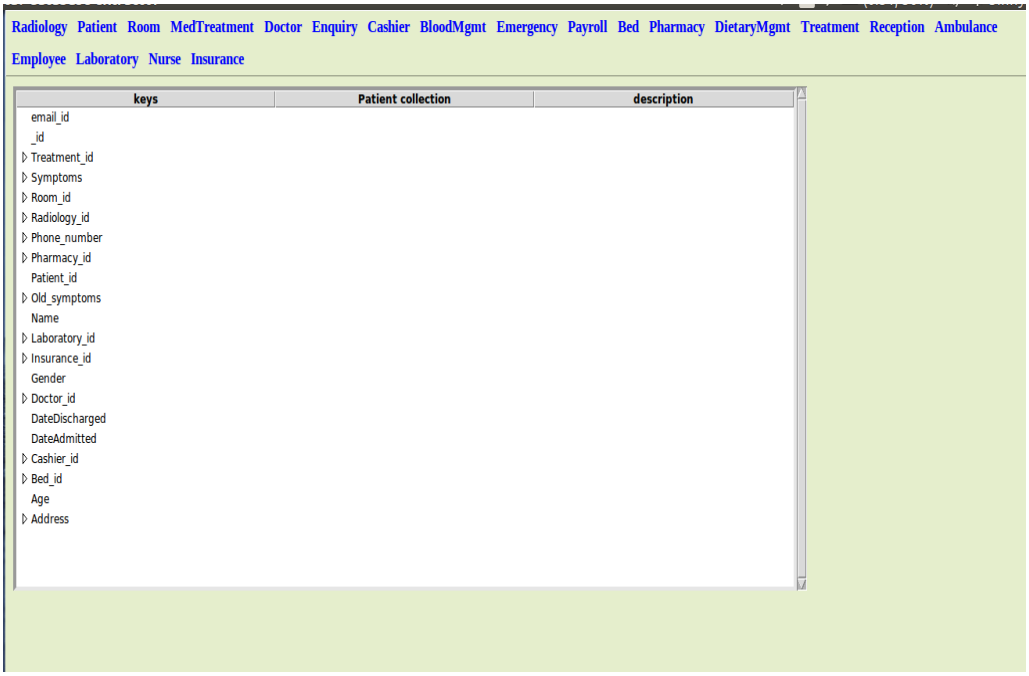

Fig 6.3: Tree View of Patient Collection

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.

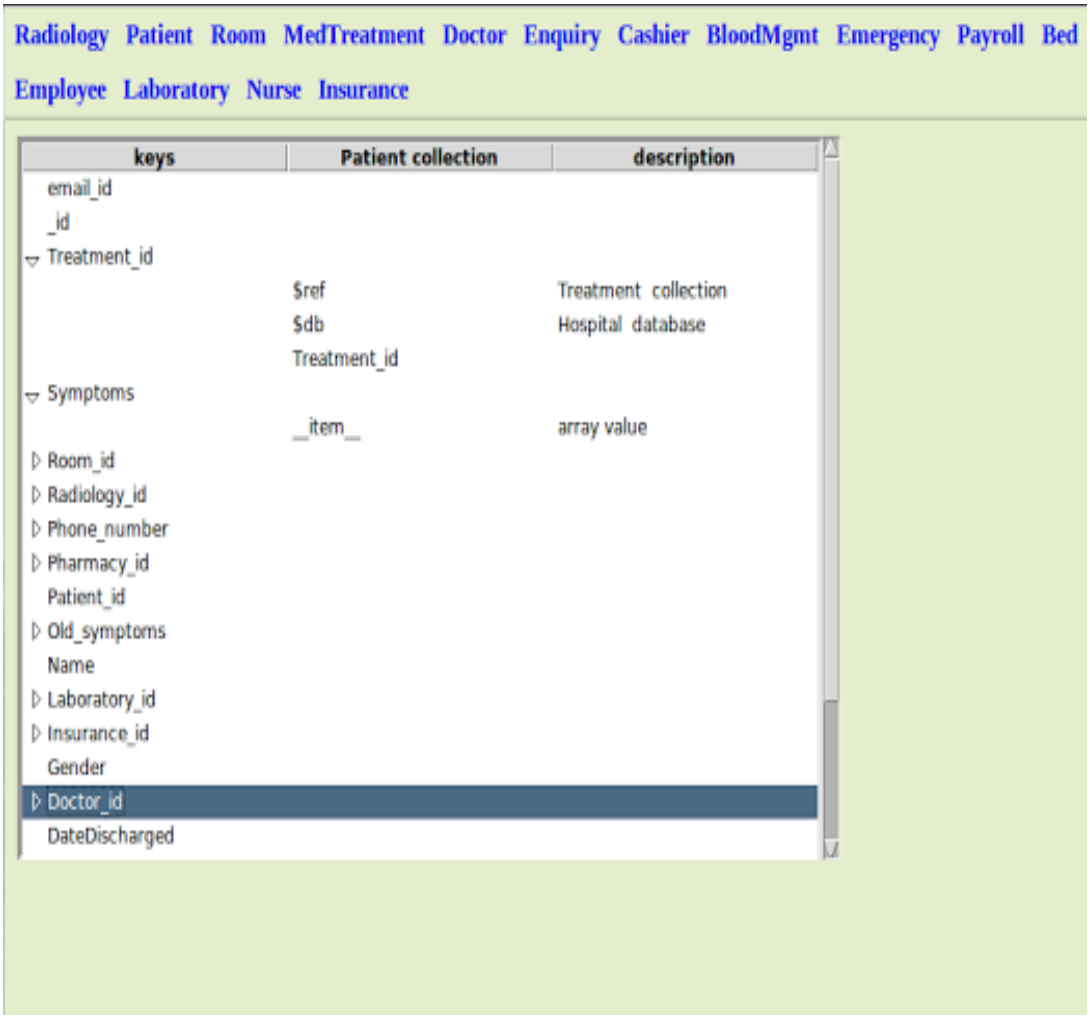

Fig 6.4: Detailed Tree View of Patient Collection

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.

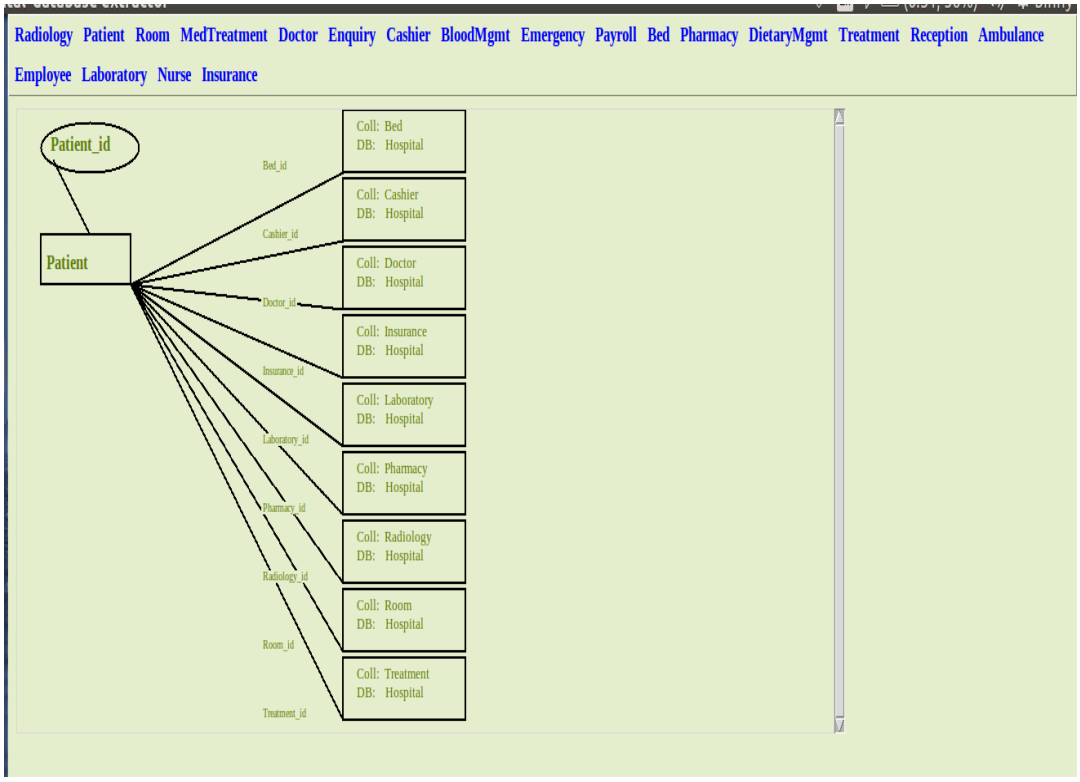

Fig6.5: Graph view of Patients information page

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.

References

[1]. Kaur, K. and Rani, R., Big Data, 2013 IEEE International Conference on, Modeling and querying data in NoSQL databases, 2013,Oct,1-7,10.1109/BigData.2013.6691765

[2]. E. F. Codd. Data models in database management. SIGMOD Rec., 11(2):112–114, June 1980.

[3]. J. Gray. The transaction concept: virtues and limitations (invited paper). In Proceedings of the seventh inter- national conference on Very Large Data Bases - Volume 7, VLDB 81, pages 144–154. VLDB Endowment, 1981.

[4]. T.Haerder and A.Reuter. Principles oftransaction oriented database recovery. ACM Comput. Surv.,15(4):287–317,Dec.1983.

[5]. S. Edlich, A. Friedland, J. Hampe, and B. Brauer.NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. Hanser Fachbuchverlag, 10 2010.

[6]. M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale out: A case study using nutch/lucene. In Parallel and Distributed Process- ing Symposium, 2007. IPDPS 2007. IEEE International, pages 1 –8, march 2007.

[7]. C.Strozzi. Nosql relational database management system. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/HomePage, July 2012.

[8]. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A.Wallach, and M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Big table: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI ’06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[9]. G. DeCandia, D. Hastorun, M. Jampani, G. Kakula- pati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon’s highly available key-value store. SIGOPS Oper. Syst. Rev., 41(6):205–220, Oct. 2007.

[10]. S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, June 2002.

[11]. D. Pritchett. Base: An acid alternative. Queue, 6(3):48–55, May 2008.

Redmond, Eric, Wilson, Jim R, and Carter, Jacquelyn. Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. The Pragmatic Programmers, LLC, 2012.

[12]. B. Bondi. Characteristics of scalability and their im- pact on performance. In Proceedings of the 2nd interna- tional workshop on Software and performance, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[13]. E. F. Codd. Data models in database management. SIGMOD Rec., 11(2):112–114, June 1980.

[14]. H. Lu, J. X. Yu, G. Wang, S. Zheng, H. Jiang, G. Yu, and A. Zhou. What makes the differences: benchmarking xml database implementations? ACM Trans. Internet Technol., 5(1):154–194, Feb. 2005.

[15]. R. Cattell. Scalable sql and nosql data stores. SIGMOD Rec., 39(4):12–27, May 2011.

[16]. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A.Wallach, and M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Big table: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[17]. R. H. Gu¨ ting. Graphdb: Modeling and querying graphs in databases. In Proceedings of the 20th Inter- national Conference on Very Large Data Bases, VLDB '94, pages 297–308, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[18]. J. Cheng, Y. Ke, and W. Ng. Efficient query processing on graph databases. ACM Trans. Database Syst., 34(1):2:1–2:48, Apr. 2009.

[19]. Pan Fan, From MySQL to MongoDB Visual China's R oad of NoSQL China [J], Programmer, 201006, 79-81

[20]. R. Angles and C. Gutierrez. Survey of graph database models. ACM Comput. Surv, 40(1):1:1–1:39, Feb. 2008.

[21]. D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of amazon s3's consis- tency behavior. In Proceedings of the 6th

Workshop on Middleware for Service Oriented Computing, MW4SOC'11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[22]. D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkarup- pan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache hadoop goes realtime at facebook. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[23]. E. A. Brewer. Towards robust distributed systems. In Symposium on Principles of Distributed Computing (PODC), 2000.

[24]. J. Dean and S. Ghemawat. Map reduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.

[25]. T. Haerder and A. Reuter. Principles of transaction- oriented database recovery. ACM Comput. Surv, 15(4):287–317, Dec. 1983.

[26]. W. Vogels. Eventually consistent. Queue, 6(6):14–19, Oct. 2008.

[27]. M. Lenzerini, "Data integration: a theoretical perspective," presented at 21st ACM SIGMOD- SIGACT-SIGART symposium on Principles of database systems, 2002.

[28]. E. Sciore, M. Siegel, and A. Rosenthal, "Using Semantic Values to Facilitate Interoperability among Heterogeneous Information Systems, "ACM TODS, vol. 19, pp. 254-290, 1994.

[29]. GUPTA, A., HARINARAYAN, V., AND QUASS, D. Aggregate-query processing in data warehousing environments. In VLDB (1995), pp. 358–369.

[30]. Alex Williams (14 November 2013)."PostgreSQL NowAvailable On Amazon's Relational Database Service".TechCrunch.

[31]. A discussion of NoSQL database "why should non- relational database" [OL] http://robbin.javaeye.com/blog/524-977

[32]. Huang Xian-li, NoSQL non-relational database development and application, Fujian PC, 201007, 30, 45.

[33]. Bruce Eckel, Thinking in Java (Second Edition) [M], Machinery Industry Press, 2002.

[34]. Nicholas C.Zakas, JavaScript Advanced Programming Second Edition [M], Posts & Telecom Press, 2010.

[35]. Brett McLaughlin, Java XML Photocopy Edition [M], Southeast University Press, 2007.

[36]. Lauriat.S.M, Ajax Architecture and Best Practices in depth [M], Posts & Telecom Press, 2009.

[37]. Dayal, U. and Hwang, Hai-Yann View Definition and Gen- eralization for Database Integration in a Multidatabase System. Software Engineering, IEEE Transactions on, SE- 10, 1984 Nov, 628-645, 0098-5589.

[38]. Shelake, V.M. and Bhojane, V.S. A Novel Approach for Multi-source Heterogeneous Database Integration. 2013 Dec,184-190,10.1109/ICMIRA.2013.42.

[39]. Jayathilake, D. and Sooriaarachchi, C. and Gunawardena, T. and Kulasuriya, B. and Dayaratne, T., Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on, A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree, 2012,Sept,106-111,10.1109/ICIAFS.2012.6419890

[40]. Tauro, C.J.M. and Ganesan, N. and Easo, A.A. and Mathew, S., Advances in Computing and Communications (ICACC), 2013 Third International Conference on, Convergent Replicated Data Structures that Tolerate Eventual Consistency in NoSQL Databases, 2013,Aug,70-75,10.1109/ICACC.2013.109

[41]. Reddy, M.P. and Prasad, B.E. and Reddy, P.G. and Gupta, A., Knowledge and Data Engineering, IEEE Transactions on, A methodology for integration of heterogeneous databases, 1994,Dec,6,6,920-933,1041-4347.

List of Research Publications

- Binny Garg, Karamjit Kaur, "Integration of Heterogeneous databases", at "International Conference on Advances in Computer Engineering and Applications" [ICACEA-2015], held on 19th-20th March, 2015, in IMS Engineering College, Ghaziabad, India.