A Novel Approach for Transformation of Data from MySQL to NoSQL (MongoDB)

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

Masters of Engineering

in

Computer Science and Engineering

Submitted By

Rupali Arora

(Roll No. 801132035)

Under the supervision of:

Dr. Rinkle Rani

Assistant Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2013

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"A Novel Approach for Transformation of Data from MySQL to NoSQL (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 **Dr. Rinkle Rani** and refers other researcher's work which are duly listed in the reference section.

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

Rupali Arora

(Rupali Arora)

801132035

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

Rinkle 28/06/13

(Dr. Rinkle Rani)
Asstt. Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by:

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide **Dr. Rinkle Rani**, Department of Computer Science & Engineering, Thapar University, Patiala, who has been very concerned and has aided for all the materials essentials for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me all the ideas on how to work towards a research-oriented venture.

I am equally grateful to **Dr. Maninder Singh**, Associate Professor and Head, Computer Science & Engineering Department, for motivation and inspiration that triggered me for the thesis work.

I will be failing in my duty if I do not express my gratitude to **Dr. S. K. Mohapatra**, Senior Professor and Dean of Academic Affairs the University, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

Most importantly, I would like to thank **Ms. Karamjit Cheema**, Lecturer, Computer Science & Engineering Department. This work would not have been possible without her encouragement and valuable guidance.

I would also like to thank the staff members and my colleagues who were always there at the need of hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Rupali Arora
(801132035)

Abstract

Relational databases have been the dominant model for the several decades, for storing and retrieving data. But in some cases relational database has shown its age and losing its importance due to its dependence on fixed schema and inability to scale well. With the development of the internet and cloud services like Google and Amazon volume of data is increasing exponentially. Data accumulated by social services such as Facebook and Twitter is more connected and semi-structured. Today's Web and mobile applications are designed to support large number of concurrent users by spreading load across a collection of servers. Traditional relational databases are facing many challenges to contend with these trends. NoSQL (Not Only SQL) databases are developed as a solution to these problems. NoSQL is an umbrella term for all data stores that do not follow RDBMS principles. They do not require fixed schema nor do they use the concept of joins. NoSQL databases are horizontal scalable, flexible to handle semi-structured data and support high availability.

NoSQL databases are classified into four classes: Key-value data store, Column-oriented data store, Document database and Graph database. These data stores are appropriate for different applications. Key-value data stores are used to handle massive loads. Column-oriented data stores are used to store large amount of data which is distributed over many machines. Document databases are used to store semi-structured data, where as graph databases are used to store data elements interconnected by many relations. This thesis provides an in-depth knowledge of NoSQL classes, and different tools available of these classes.

With the social media posts and multimedia, the need of unstructured data storage has grown rapidly. MongoDB document databases are used to overcome these problems. MongoDB stores the data in the form of BSON documents. In this work an approach has been proposed for transformation of data from MySQL relational database to MongoDB document database. Further, the proposed approach has been implemented using NetBeans IDE and Pentaho.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Database is organized collection of data and the term database management system is used to describe a variety of software systems that allows database functions of storing, retrieving, adding, deleting and modifying data. There are different types of systems as well as models that perform database functions in different ways. The fundamental differences of more significant models are discussed here. Traditional relational databases have been around for decades. Recently, many new NoSQL systems are emerging, which have renewed interest in non-relational storage models. These storage models have been discussed in the following sections.

1.1 Relational Model

Relational model have been used as the most common data storage model for several decades. The term was originally defined by Edger Codd at IBM Almaden Research Center in 1970 [1]. The software which implements relational model is known as relational database management system or RDBMS. Modern relational database management systems are based on the concept of Codd's relational model. Examples of RDBMS are MySQL, Oracle, PostgreSQL , Microsoft SQL server.

In relational database management systems data is organized into tables consisting of rows and columns. In Codd's model the table is referred to as relation, columns are referred to as attributes and rows are referred to as tuples. The relation is basic element of relational model. Relation consists of set of tuples having the same attributes. One tuple represent an object and gives information about that object. To distinguish all the tuples in a relation, the set of one or more attributes are chosen as a primary key. To relate two relations, relational database use the concept of foreign key. A foreign key in a relation refers to the primary key in other relation. Compared to primary key, foreign key need not be unique.

An example of the relational model represented by relations also called table is shown in Figure 1.1. This relational database stores information about customers, orders and the orders made by customers. Underlined attributes in Figure 1.1 are primary keys.

Customer					
Cid	**First Name**	**Last Name**	**Street**	**City**	**State**
101	Vedaang	Mittal	Connaugh place	New Delhi	Delhi
102	Veni	Goyal	Sec_17	Chandigarh	Punjab
• •	• •	• •	• •	• •	• •

Order	
Oid	**Total Price**
151	1200 rs.
160	1560 rs.
• •	• •

Last_Order	
Cid	**Oid**
101	151
102	160
• •	• •

Items	
Itemid	**Oid**
56234	151
16356	151
15016	160
70023	160
• •	• •

Figure 1.1: Example of Relational Database

To retrieve the data from database, relational model has defined SELECT operations that includes projections and joins. To modify the relations INSERT, DELETE, UPDATE operations are defined. To perform these operations, RDBMS uses a declarative query language known as SQL (structured query language). SQL is Structured Query Language which is a query language for storing, manipulating and retrieving data stored in relational database. It is an ANSI standard but there are many different versions of the SQL language. SQL also provides easy method to create views and set permissions on tables, procedures, and views.

Relational databases have been used for storing and retrieving data in computer industry since a long time. But in some cases relational databases has shown its age. One issue that has been widely debated is the object relational impedance mismatch [2]. "Object relational impedance mismatch" is a set of conceptual and technical problems experienced when relational database is used to store the objects from programs written by object-oriented programming languages. This is because relational model stores data in form of rows and columns. But object data model is not constrained to keep data in rows and columns. Instead, developers created a definition

known as template that completely describes a certain class of information. Every record (object) is instance of class. Class definition also includes methods, which act upon the data described by the class. There is no analogous construct in relational model.

A relational database also suffers from some runtime problems in very common and important scenarios. Firstly, relational database shows very poor performance characteristics for sparse tables. Sparse tables are tables where many rows have no values for many columns. Sparse table data is in semi-structured form but relational model is built to handle data in structured form i.e. data with defined and complete schema. The problem is that, as Stefano Mazzocci of Apache and MIT research fame states it, *"The great majority of the data out there is not structured and there is no way in the world you can force people to structure it."* [3]. Secondly, relational databases shows poor performance for the data sets those are naturally ordered in form of networks. Like the data sets generated in semantic web, content management, artificial intelligence, social networks and bio-informatics. In relational model such data sets are stored on multiple tables and costly joins operations are needed to relate them. One join corresponds to a "jump" along an edge between two nodes in a network.

Recently, due to cloud services like Google, Amazon data is growing exponentially. Corporate database accumulates terabits of data per day. Due to social networks like Facebook and Twitter data is getting more connected and semi-structured. The relational databases are facing many new problems everyday to confront these challenges. Few problems with relational databases are listed here.

- **Scalability:** The scalability of a system is its capability to cope with growing workload [4]. There are two types of scaling: vertical scaling and horizontal scaling. Vertical scaling ("Scaling up") means increasing the capacity of system nodes, that can be achieved by using more powerful hardware (more memory, faster processor, and larger disks). In horizontal scaling ("scaling out") more nodes are added. As RDBMS follow ACID and rich query model. The best way to achieve this is to have dataset on a single machine that is RDBMS having vertical scaling. But an organization finds it cheaper and feasible to scale out by adding more nodes rather than investing on a single node.

- **Complexity:** RDBMS performance falls off as it normalizes data. As there will be more tables, table joins and thus more internal database operations for query implementation, when the database starts to grow into terabytes, the system slows down.
- **SQL:** SQL can be used conveniently with structured data. However, it is difficult to use this language with other type of information like semi-structured data, because it is designed to be structured.
- **Object-relational impedance mismatch:** The object-relational impedance mismatch makes it very difficult and time consuming to fit an object oriented object into relational table.
- **Fixed schema:** A problem with relational databases is schema evolution. The most powerful characteristics of relational databases that are requirement of fixed data model and referential integrity are not according to needs of changing business requirements. Scaling and semi-structured data storage are the requirements of today's businesses.

1.2 Non-Relational Databases

Although relational databases have been widely used in industry for many years, there exist some non-relational databases also. Three of these non-relational databases: Object-Oriented databases, XML databases and NoSQL databases. Non-relational databases are discussed in the following section.

1.2.1 Object-Oriented Databases

Object oriented model replaced the relational model in 1985 as the dominant solution of structured data storage [5]. The object oriented model is based on the collection of objects. The main purpose of the object oriented model is to bridge the gap between real world and the data model. A single language interface between the Data Manipulation Language (DML) and programming languages overcomes the impedance mismatch. This eliminates many of the inefficiencies that occur in mapping a declarative query language such as SQL to an imperative language such as "C".

Unlike relational model, object oriented model is not value oriented. Objects are real world objects and objects belong to class. A class contains both object attributes and operation definitions. In object oriented model, operations are known as

methods. Methods and attributes can be defined as public or private. Private methods are visible to only object itself and public methods are visible to other objects also. Attributes and methods can also be inherited by other classes publically or privately. Data transmission between objects can be done by passing messages to the methods. Due to difficulties of the data transmission, object oriented databases have never been in a mainstream use and finally these were subsumed under the relational databases and object-relational databases came into the existence. These databases have object-oriented front end on a relational database. An application interfaces with this type of database as though the data is stored as objects. However, the system will convert the object information into data tables with rows and columns and handle the data the same as a relational database. Similarly, when the data is retrieved, it must be re-assembled from simple data into complex objects. Data representation in Object-Oriented databases is shown in Figure 1.2.

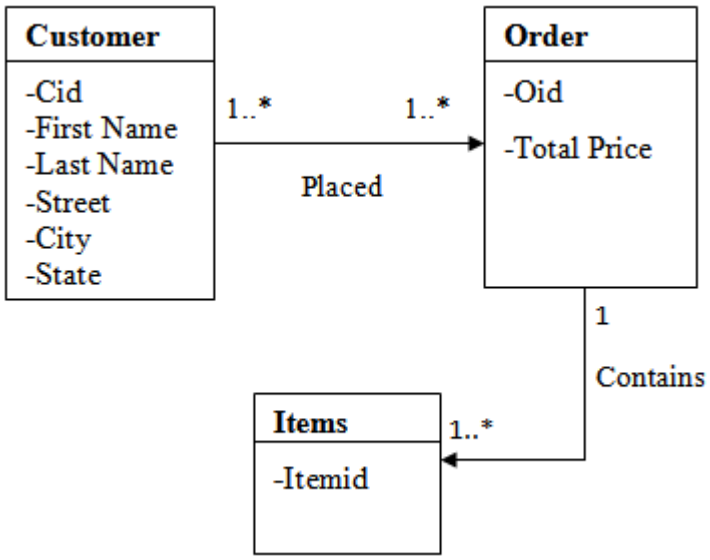

Figure 1.2: Representation of Data in Object-Oriented Database

1.2.2 XML Databases

With the development of Internet, there was a need of standard common data format for data exchange between applications. XML 1.0 and XML 1.1 solved this problem. XML is used in web-based and business applications to transport the data stored in the relational databases. The software systems which are used to manage the XML data are known as XML databases. XML databases are non-relational databases

with query languages XQuery, XPath or XUpdate. The major classes of XML databases are as follows [6].

- XML-Enabled Database (XEDs) - To store the XML data in these databases, each XML document is decomposed and its data are stored into tables. But these databases were not capable to handle documents of large sizes as decomposing and recomposing huge documents meant a high number of join operations, leading to unacceptable performance, both in retrieving documents and in querying them.
- Native XML Databases (NXDs) – These databases were specially developed for storing XML documents. These databases adopts XML data model for storing XML data. In these databases, hierarchy and ordering of nodes of XML documents can be maintained efficiently. This improves the performance in handling large XML documents.
- Hybrid XML Databases (HXD): These databases can be treated as either a Native XML Database or as an XML Enabled Database depending on the requirements of the application.

However XML and Object-oriented databases are non-relational database but finally they are subsumed under the relational databases. XML is integrated with relational databases to enable the storage, retrieval and update of XML documents. Object-Oriented databases have subsumed under relational databases and Object-Relational databases came into existence. So, these databases are not true non-relational databases. But NoSQL databases discussed below are truly non-relational databases.

1.2.3 NoSQL Databases

The term "NoSQL" was first used by Carlo Strozzi [7] in 1998 for his RDBMS, Strozzi NoSQL. Since his database solution does not use SQL, so he used the term NoSQL to distinguish his database solution from other RDBMS solutions which use SQL. Recently the term NoSQL(Not Only SQL) is used as an umbrella term for all databases and data stores that do not follow RDBMS principles and do not use SQL(Structured Query Language) as its query language.

In 2009 Stefan Edlich [8] describes NoSQL database as "*Next Generation Databases mostly addressing some of the points: being non-relational, distributed,*

open-source and horizontal scalable. The original intention has been modern web scale databases".

Dynamo, developed by Amazon [9], Google's BigTable [10] or Hadoop [11] used by Facebook, are examples of NoSQL databases. NoSQL main goals are to provide high-scalability, high availability and high write throughput.

1.2.3.1 Need of NoSQL Solutions

Relational databases have been used since the 1980s for general data storage in web and business applications with millions of reads but few write requirements. With the advent of Web 2.0 applications and social-networking applications, that required million of user reads and writes, storage of information, support and maintenance, have become the biggest challenge. Facebook and Twitter accumulate terabytes of data everyday for millions of its users [12]. Google have to store large amount of data in the form of web pages and links. In 2008, it already passed the mark of more than 1 trillion unique URLs [13]. These kinds of services are likely to grow massively in next few years [14].

Relational databases have increasing problem to cope with these trends [15]. Relational database management systems support SQL query language, Strong consistency and Referential Integrity. They are designed to handle few write requests over a day and stores data on a single instance on a server. As the database grows, more tables, table joins, keys and internal database operations needed for implementing queries and then the system slows down.

Technologies such as NoSQL were developed to deal with large scale needs and storage capacity limitations of traditional relational databases. NoSQL (Not Only SQL) databases are non-relational, distributed and horizontal scalable database. They do not require fixed table schema nor do they use the concept of Joins. It does not support SQL query language. NoSQL was inspired and created based off Lotus notes, a program that was co-created by Lotus founder Mitch Kapor and Microsoft chief architect Ray Ozzie as a personal productivity tool [16]. Unlike relational database, NoSQL does not store information as normalized tables, but stores as documents. Documents are stored in variant of JavaScript object notation which is similar to XML.

A key feature of NoSQL systems is "Share nothing" horizontal scaling-replicating and partitioning data over many servers [17]. Due to this feature, NoSQL systems can support a large number of simple read/write operations per second.

NoSQL does not provide ACID (Atomicity, Consistency, Isolation, Durability) guarantees but follows BASE. BASE is an acronym for Basically Available, Soft state and Eventual consistency [18]. Basically available means that an application works mostly. Soft-state means data is not consistent all the time but will be in eventually consistent state.

1.2.3.2 CAP Theorem

In 2000, Professor Eric Brewer [19] formulated a theorem, called CAP Theorem. The Theorem states that a distributed system can have only two out of the following three properties: Consistency, Availability and Partition-tolerance. The proof of this theorem was done by Gilbert and Lynch in 2002 [20].

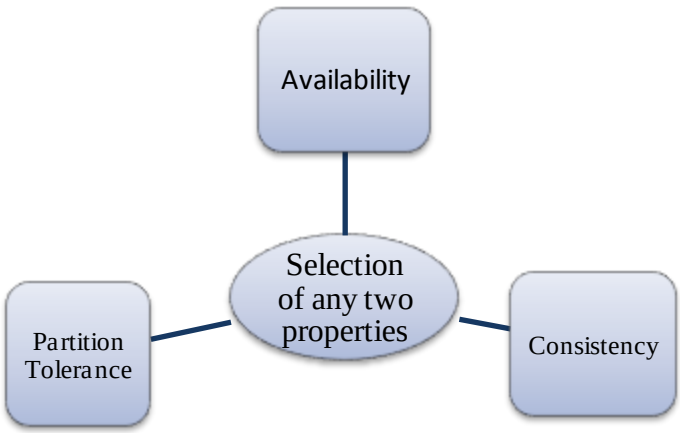

Figure 1.3: CAP Theorem

The CAP Theorem can be summarized as follows:

- **Consistency:** A system is in consistent state if after every write, every client requesting for reads from the database is guaranteed to get the most recent value.
- **Availability:** Availability means that a system is designed in such a way that allows it to continue operation even after some hardware or software failure. In other words write requests from clients are always accepted regardless of current state of the system.

- **Partition tolerance:** It means that system accepts read and write requests from clients for the nodes that are unreachable. This situation can occur when the nodes are inaccessible on the networks or the networks links between the nodes are interrupted. But in case of partition tolerance client can't see the partition.

1.2.3.3 Advantages of NoSQL Databases over Relational Databases

The NoSQL databases have many advantages over relational databases because it allows horizontal scalability and applications can scale to new levels. NoSQL solutions are built for the cloud services like Google, Amazon and can be easily distributed. There is no need of database administrator (DBA) and complex SQL queries. Few advantages of NoSQL solution over relational databases are as follows:

- **Open Source:** NoSQL provides open source products to developers. Open source products are more reliable, secure and faster to deploy as compared to proprietary alternatives.
- **Elastic scaling:** Unlike relational databases NoSQL databases are horizontal scalable. The capacity of servers can be increased by simply adding more servers. No downtime is required.
- **Flexible data models**: NoSQL databases are schema-less, thus there are no data modeling restrictions. Records can have variable number of fields that can vary from record to record.
- **Big Data:** Due to social networks and cloud services data is growing exponentially. Relational database management systems have problems to cope with these trends. NoSQL databases can handle volume of "Big Data".
- **No DBA requirement**: Due to automatic repair, data distribution and simple data model NoSQL databases require less administration and tuning
- **BASE instead of ACID:** NoSQL databases does not implement ACID, instead they support BASE and emphasize on performance and availability.

1.2.3.4 Challenges for NoSQL Databases

The promise of the NoSQL database has generated a lot of enthusiasm, but there are many obstacles to overcome before they can appeal to mainstream enterprises. Here are a few of the top challenges [21].

1. Maturity: RDBMS systems have been around for a long time. For the most part, RDBMS systems are stable and richly functional. In comparison, most NoSQL alternatives are in pre-production versions with many key features yet to be implemented.

2. Support: Enterprises want the re-assurance that if a key system fails, they will be able to get timely and competent support. All RDBMS vendors go to great lengths to provide a high level of enterprise support. In contrast, most NoSQL systems are open source projects and although there are usually one or more firms offering support for each NoSQL database, these companies often are small start-ups without the global reach and support resources.

3. Analytics and business intelligence: Businesses mine information in corporate databases to improve their efficiency and competitiveness and business intelligence (BI) is a key IT issue for all medium to large companies. NoSQL databases offer few facilities for ad-hoc query and analysis. Commonly used BI tools do not provide connectivity to NoSQL.

4. Administration: The design goal for NoSQL may be to provide a zero-admin solution, but the current reality falls well short of that goal. NoSQL today requires a lot of skill to install and a lot of effort to maintain.

5. Expertise: There are millions of developers throughout the world who are familiar with RDBMS concepts and programming. In contrast, almost every NoSQL developer is in a learning mode. This situation will address naturally over time, but for now, it is far easier to find experienced RDBMS programmers or administrators than a NoSQL expert.

6. Standardization: Each NoSQL database has its own set of APIs, libraries and preferred languages for interacting with the data they contain. With an RDBMS, it is trivial to get data out in any format and programming language. Choice of a NoSQL database might limit to one or a handful of programming languages and access methods.

1.3 Objectives

Objectives of this thesis are as follows.

1. To study in detail relational and various NoSQL databases and compare them.
2. Installation and hand-on experience on tools available for Document-based data stores.

3. To propose a technique for transformation of data from relational database to document-based NoSQL database.
4. Implementation of the proposed technique using NetBeans Java IDE and analyze the results.

1.4 Thesis Outline

The rest of the thesis is organized in the following order.

In **Chapter 2** classification of NoSQL solutions have been discussed. Popular tools available of these classes are also discussed in this chapter. In **Chapter 3** problem statement and methodology used to solve it has been discussed. In **Chapter 4** document-oriented databases are explained and various available popular document databases are compared. In **Chapter 5** the design and implementation of proposed approach for transformation of relational databases to document databases has been discussed. In **Chapter 6** conclusion and future scope of the presented work has been outlined.

Chapter 2

Classification of NoSQL Solutions

NoSQL solutions are identified into four Classes: Key-Value data stores, Column-oriented data stores, Document databases and Graph databases [22]. These data stores work differently and used for different applications. Key-value data stores uses map/dictionary data model where key to value persistent map is used for value retrieval. These data stores are designed to handle massive data loads and their applications include content caching. Column-oriented data stores are based on Google's Big Table and used to store very large amount of data distributed over many machines. Document databases used to store semi-structured data and used in web-applications, where as graph databases stores data in graph like structures and used in social networking or where data elements are more interconnected with an undetermined number of relations between them. The detailed description of the classes and different tools available of these classes is given in the following sections.

2.1 Key-value Data Stores

A key-value database or key-value store is similar to a relational database with rows, but only two columns (key and value) [23]. So, these storage systems are basically associative arrays, consisting of key-value pairs. Each key is unique and used to retrieve the values associated with it. The features of key-value stores are as follows.

- The query speed of these data sores is higher than relational databases and used for faster look-ups.
- Schema-less, means the value can be anything (objects, lists or hashes again). Because of this, there is no need of fixed data model.
- Insertion and deletion operations – The values corresponding to key can be inserted or deleted.

Key-value data stores use a data model similar to the popular memcached (distributed in-memory cache), with a single key-value index for all the data. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering [24].

Besides the data-model and the API, modern key-value stores favor high scalability over consistency and therefore most of them also omit rich ad-hoc querying and analytics features (especially joins and aggregate operations are set aside). Often, the length of keys to be stored is limited to a certain number of bytes while there is less limitation on values ([25], [26]).

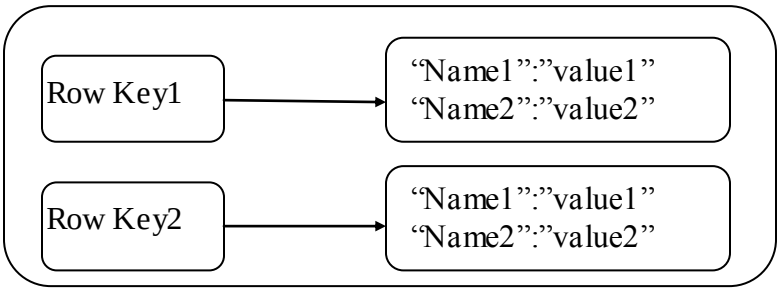

Figure 2.1: Representation of Key-value Data Store

Figure 2.1 shows the representation of the key value data store. Data is stored in the form of keys and values, where keys are simple objects and values can be lists, sets or hashes again. Unique row key is used to distinguish each row.

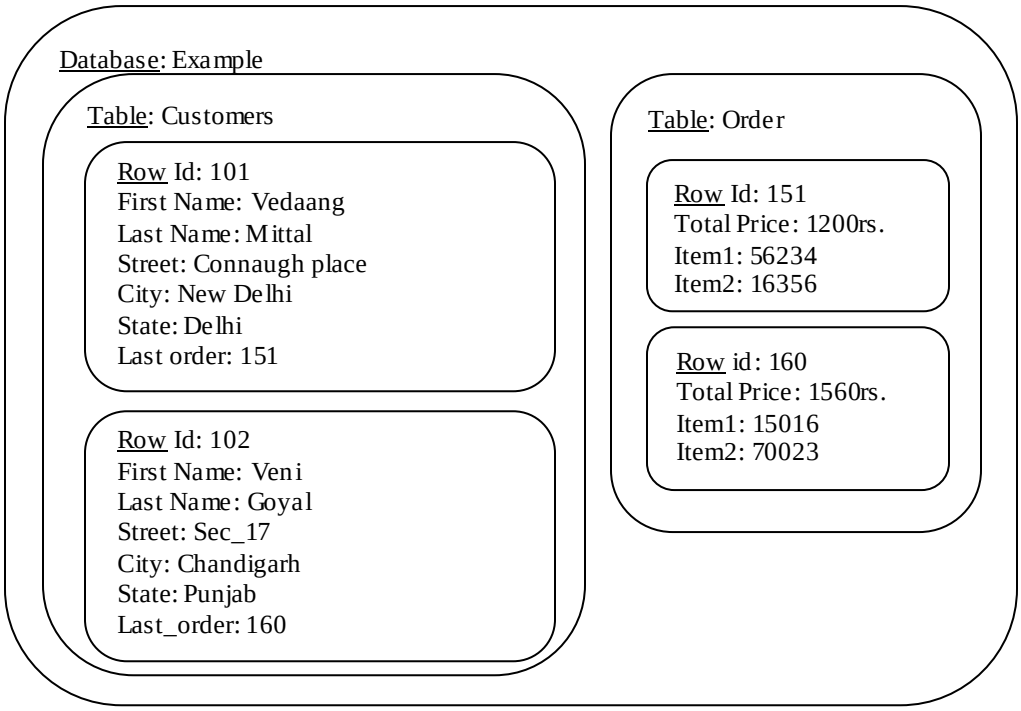

Figure 2.2: Example of Key-value Data Store

Figure 2.2 shows an example of key-value data store. Nomenclature of tables and rows are used. Rows contain collection of keys and values, which vary from row to row. RowID is used to distinguish rows. It has two tables named customers and

order. Customer table has two rows with RowID 101 and 102 and similarly order table has two rows with RowID 151 and 160.

These data stores are used in schema evolving applications, e.g. they can be used for storing market data from stock exchanges. Riak [27], Voldemort [28], Redis [29], Tokyo Cabinet [30] are key-value data stores. They offers an efficient and highly scalable storage solution, but at the cost of limited querying interface.

2.1.1 Project Voldemort

Voldemort is an open-source Key-value data store. It is developed by LinkedIn and currently in use at LinkedIn [28]. It is written in Java and is available under the Apache 2.0 license. Thrift API is used to access the database; there are native bindings to high-level languages as well that employ serialization via Google protocol buffers [31]. Voldemort provides Multi-version concurrency control (MVCC) for updates. In this data store replicas are updated asynchronously, so it is eventually consistent and does not guarantee consistent data. In voldemort data is automatically replicated over multiple servers and partitioned automatically so that each server contains only subset of data.

Voldemort supports Berkeley DB and Random Access File storage, means it can store data in RAM but plugging in a storage engine is also permitted. In addition to simple scalar values voldemort also supports lists and records.

2.1.2 Riak

Riak is a fully distributed, scalable key-value data store developed by Basho [27]. It was open-sourced under Apache2 license in August, 2009. Riak is written in Erlang. It provides Multi-Version Concurrency Control (MVCC) for updates. In Riak, vector clocks are used as version control mechanism, which are assigned when values are updated.

Riak has both the features of key value and document databases. Like document database, in Riak objects are stored and fetched in JSON (JavaScript Object Notation) format and can have multiple fields and can be grouped into buckets (collections in document databases). But Riak does not have query mechanism of the document database. Only primary key lookup can be done. It also supports links which are relationships between objects. Links simplifies traversal requests and it also supports MapReduce in both Erlang and JavaScript.

2.1.3 Redis

Redis is an open source, BSD licensed key value data store [29]. Redis is written in ANSI C. It stores the data in the form of keys and values, where values associated with keys can be lists, ordered lists and sets. Redis load the whole document into main memory. So, all operations are run in memory but periodically save the data on disk asynchronously. Due to memory operations its performance is high. Redis supports master-slave replication to scale reads and Sharding to distribute writes. This data store also offers shell for simple interaction.

2.1.4 Tokyo Cabinet

Tokyo Cabinet is an open source key value data store [30]. It was developed by Mikio Hirabayashi in Japan mainly for Japan's largest SNS site mixi.jp, it has been a very mature project yet [32]. Tokyo Cabinet is back-end server dealing with all of the data structures like B-Trees and Hash Tables. Tokyo Tyrant is networking interface that provides remote access to the data stored in Tokyo Cabinet. They support ACID transactions and binary array data types. They also support master/slave or dual master replication and have no support for automatic sharding.

2.2 Column-family Data Stores

Column-family data stores are similar to key-value data stores but here the data is stored by columns. Columns of similar data are stored in a same file on disk known as column family. A column family contains row key similar to row key in key value store. A row key contains super-column or column. Super column is a column that contains other columns but not super-columns. A column is a tuple of name and value. Figure 2.3 shows the representation of column-family data stores. Compared to relational databases keyspace corresponds to database, column family to table and row key to primary key, column name/key to column name and column value to column value.

The main goal of column-family data stores is to store and process large amounts of data distributed over many machine. Here row key is used as “shard key”. Sharding [33] is a technique which involves partitioning of data by rows across a number of distributed servers. They typically split by range rather than a hash function; it means there is no need to traverse every node for queries on a range of values. “Column groups” are used to distribute columns of a table over multiple

nodes. Column groups are simply a way to indicate which columns are best stored together. Horizontal and vertical partitioning can be used simultaneously on the same table. For example, if a customer table is divided into three column groups (say customer name/address, log-in information and financial details). Then for the purpose of sharding the rows by customer Id, each of the three column groups id treated as separate table. The column groups for one customer may or may not be on the same server. Rows correspond to documents. Like documents, they also can have variable number of attributes (fields) and the attribute name is unique. These rows are grouped into collections and attributes of individual row can be of any type. These extensible record stores have been motivated by Google's success with Big Table where Big Table is NoSQL data store introduced by Google in 2007 [17] . These data stores are mainly used in read/write intensive applications like social-networking. The leading technologies in this area are Google's Big Table [10], HyperTable [34], Cassandra [35] and HBase [36].

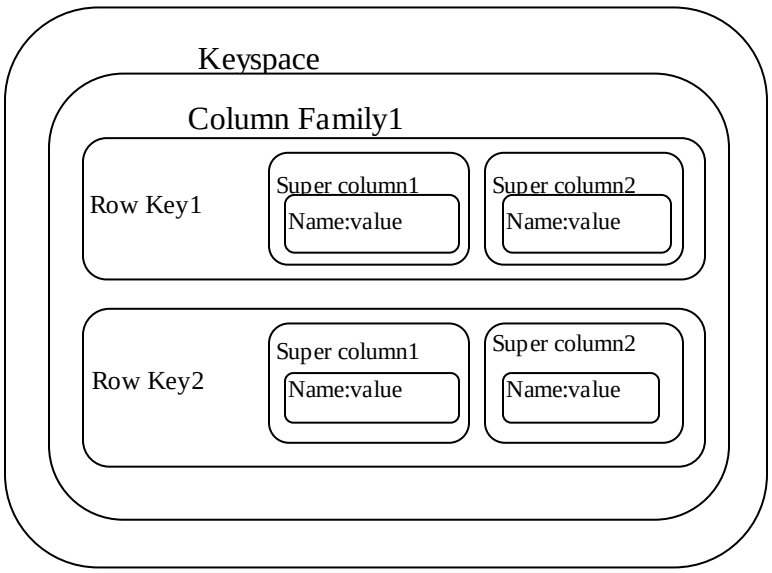

Figure 2.3: Representation of Column-family Data Store

An example of column-family data stores is shown in Figure 2.4. As can be seen in this example, there are two column families in example keyspace named customer and order. Customer column family has two row keys 101 and 102 and order column family has two row keys 151 and 160. Row keys 101 and 102 have three super columns: Name, Address and Orders and row keys 151 and 160 has two super columns: Pricing and items. These super columns have columns of name and value tuples.

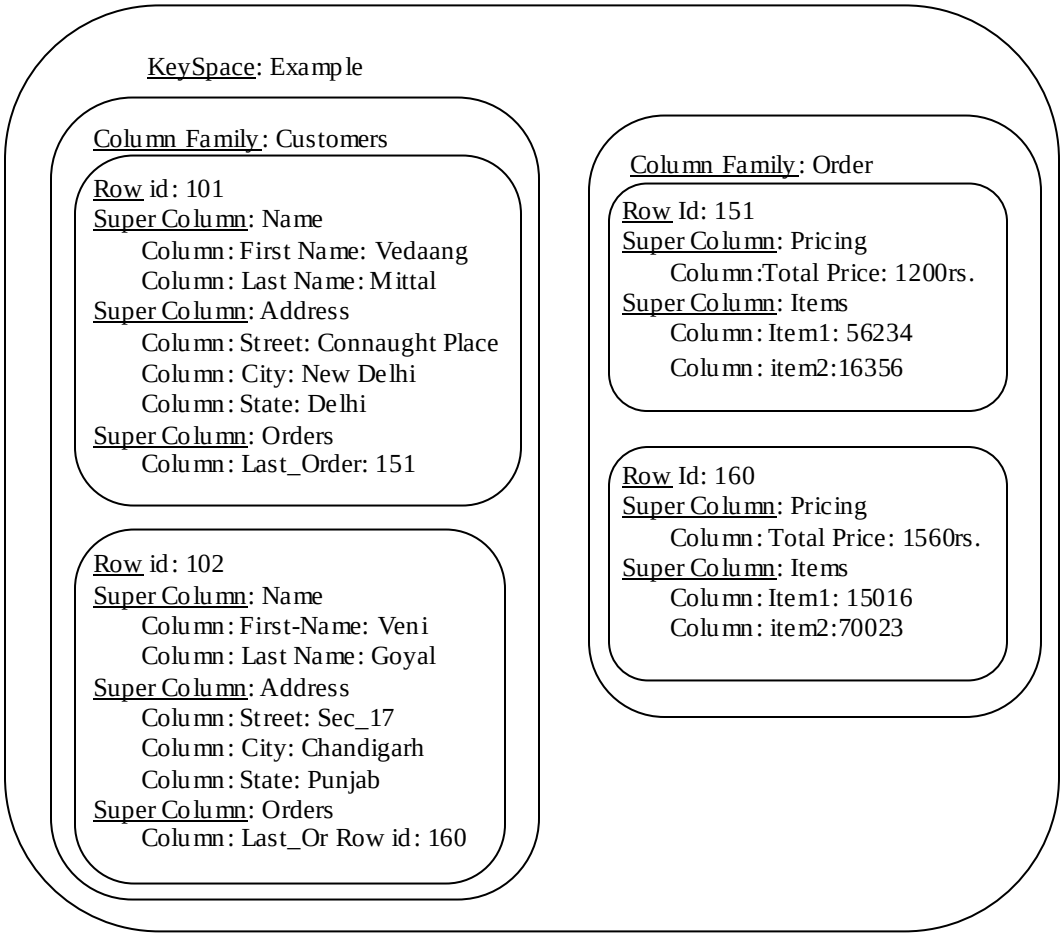

Figure 2.4: Example of Column-family Data Store

2.2.1 Google's Big Table

Bigtable as described by Google, is "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers" [10]. Big Table was designed to achieve distributed storage, high availability and high performance. Big Table provides structured storage of data like traditional relational databases but adopts a different model from relational databases. The data model of Big Table can be inferred as multidimensional sorted map. The data structure of Big Table is collection of key-value pairs. Big Table consists of column families. Column families consist of rows. Rows consist of column with key-value pairs. Timestamp are used to version the value of each column key.

Google Big Table uses Google File System, Chubby Lock Services and SST table technologies. Many projects at Google store data in BigTable, including web indexing, Google Earth and Google Finance [10].

2.2.2 HyperTable

HyperTable is an open source column-oriented data store developed by Zvents Inc. in January 2009 [34]. HyperTable was designed to solve the problems of scalability. HyperTable delivers maximum efficiency and optimum performance. It is based on the design of Google (Google's Big Table) to meet the scalability requirement. HyperTable can be deployed on the top of the Hadoop HDFS or cloud store KFS file systems. It supports SQL like query language known as HQL. It is written in C++. HyperTable has tales consists of rows having unique rowID. Each row consists of column families that can have number of column qualifiers. Each of these column qualifiers defines a unique column value. Replication and partition of tables over servers is done using key ranges.

2.2.3 HBase

HBase is an open-source, distributed column-oriented data store [35]. It is based on HDFS (Hadoop Distributed File System) [37]. It is written in JAVA. It supports ACID transactions. Thrift interface is used to access the HBase and it also supports REST calls. The HBase data model is based on BigTable data model [10]. Each HBase table has a name. Rows have RowID, which is user defined array. Tables are organized into regions, which are stored separately. Each region has a contigious RowID defined by (first row, last row). Columns are organized into column family.

HBase can handle both structured and semi-structured data naturally. It can store unstructured data too, as long as it is not too large [38]. It has dynamic and flexible data model and does not constrain the data types and kind of the data to be stored. For one column, it can store integer in one row and string in other row.

2.2.4 Cassandra

Facebook is one of the most popular social networking websites. It is used by millions of people to connect with friends and relatives. Facebook applications required a data store that can manage billions of writes per day. To meet these requirements, Facebook engineer Prashant Malik and Dynamo's original author Avinash Lakshman have designed a distributed, scalable, column-oriented datastore that combines Google Big Table's structured data model with Amazon Dynamo's eventually consistent, decentralized storage model.

Cassandra is an open source data store with flexible schema [35]. It does not require to design a schema at first and add or delete fields is very convenient [32]. Cassandra is written in JAVA and Thrift API is used to access Cassandra. Other APIs such as REST APIs can also be used. Cassandra can also have super-columns with in column families that have related columns. It provides another level of grouping.

Cassandra is distributed data store which is composed of lots of database nodes. Data is replicated on these nodes based on eventual-consistency. To achieve scalability only nodes need to be added. All nodes in a cluster are same. Cassandra uses an order hash index which gives benefit of both hash table and B-Tree index. Cassandra supports rich data model and powerful query language.

2.3 Document Databases

A document database is used to store, retrieve and manage semi-structured data. In this database, data is stored in the form of documents. The representation of document databases is shown in Figure 2.5. The format of document can be XML, YAML, and JSON. The documents are stored into collection. The documents do not need to have static schema that is documents could have completely different fields. Any number of fields can be added to the documents without wasting space by adding same empty fields to the other documents in a collection. So, document databases are easy to use and dynamic data can be easily created.

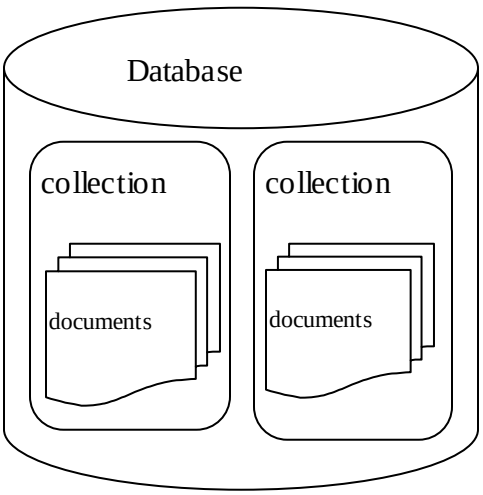

Figure 2.5: Representation of Document Database

Unlike the key-value stores, these systems generally support secondary indexes, multiple types of documents (objects) per database, and nested documents or lists [17]. Like other NoSQL systems document data stores do not provide ACID

transactional properties. Document database is not concerned about high performance read and write concurrent, but rather to ensure that big data storage and good query performance [32]. Most popular document databases are CouchDB [39], MongoDB [40], Terrastore [41] and ThruDB [42].

Figure 2.6 shows an example of the document databases. Document database contains collections. Collections contain JSON objects, referred to as documents, each of which has a schema-free set of properties and values. Values can be embedded document or reference to another document. As can be seen in this example, sample database has two collections customer and order. Customer has attributes id, first name, last name, address and orders, where address is embedded document and Orders refers to another document. Order collection has attributes id, total price, item1 and item2.

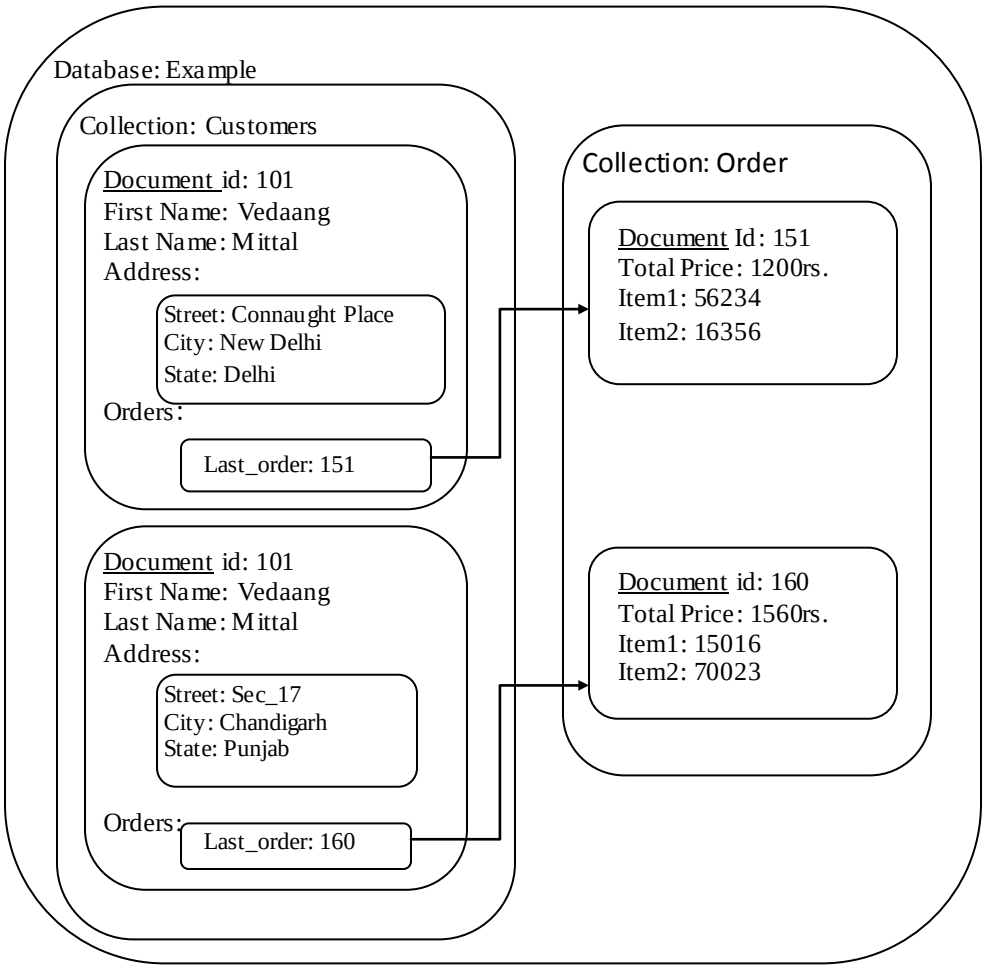

Figure 2.6: Example of Document Database

2.3.1 CouchDB

CouchDB (Cluster of Unreliable Commodity Hardware Data Base) is schema-less, fault-tolerant, distributed document database [39]. It is maintained by Apache Software Foundation and written in Erlang. CouchDB is schema free, it means in this data-store data is stored in arbitrary JSON format. To access the CouchDB data store RESTful HTTP/JSON API is used.

To manage concurrent access to the documents, CouchDB uses Multi-Version Concurrency Control (MVCC). It means documents are versioned using SequenceID. If someone fetches the document for updates, CouchDB will notify the application. Then application combines the update or overwrites the update with its update. So the file on disk is always in consistent state. CouchDB is already used in research for data management and workflow management at CERN [43].

To query the database "views" are used. Views are of two types: Temporary views and permanent views. Temporary views are loaded into memory and permanent views are stored on disk [13]. Views are basically map/reduce functions, implemented in JavaScript, that process the data stored and return the results. CouchDB keeps these views updated by running new inserts through map/reduce functions. CouchDB supports asynchronous replication for scaling. It has interfaces with many programming languages like JAVA, C, PHP, LISP, Python that convert Native API calls into RESTful calls.

2.3.2 MongoDB

MongoDB is a GPL open-source document database developed by 10gen in 2008 [40]. It is written in c++. Language bindings for many programming languages such as C, C++, Erlang, JAVA, Python, Ruby and many more have been implemented to access the MongoDB. Documents are stored in binary JSON format called BSON.

Unlike CouchDB which provides Multi-Version Concurrency Control (MVCC) for documents, MongoDB provides atomic operations on fields. MongoDB supports dynamic queries with automatic use of indices. Unlike CouchDB, It is also possible to create indices on specific fields in documents. But indexes are used where collections have the number of reads greater than the number of writes.

MongoDB supports Master-Slave asynchronous replication to have redundancy and automatic failover. Due to asynchronous replication it gives high performance but on crash some updates are lost. It also supports automatic sharding to

have horizontal scaling. Sharding is done on per collection basis not on whole database.

2.3.3 Terrastore

Terrastore is open-source, distributed database [41]. It is based on Terracotta that is an industry proven and clustering technology. It uses an Apache 2.0 license. It is written in JAVA and can be accessed through HTTP protocol. It provides advanced scalability and elasticity features without sacrificing consistency. It provides per-document consistency feature that is latest value is obtained. It is schema-less, it means data is stored in JSON format. It is easy to work with Terrastore, as fully working cluster can be easily installed by using only few commands and there is no need to edit any XML file.

2.3.4 ThruDB

Thrudb is a set of simple services built on top of Facebook's Thrift framework that provides indexing and document storage services for building and scaling websites [42]. Its main purpose is to provide flexible, fast and easy-to-use services to the developers. It has multiple storage backends like BerkeleyDB, Disk, MySQL, and S3. Thrudb provides web scale data management by having various services.

Thrudoc is a simple key value storage system designed to work on a number of underlying storage backends, such as files on disk or S3 records [42]. Thrucene provides lucene based indexing for searching data from document. Throxy provides services of partitioning and load balancing. Thruqueue provides persistent message queue services.

2.4 Graph Databases

The websites like Facebook, Twitter and LinkedIn are the consequences of web 3.0. These websites accumulate terabits of data every day and the data accumulated by these websites is more connected and semi-structured. The use of relational databases leads to problems because of data modeling (fixed-schema requirement) and horizontal scalability constraints. Technologies like graph database were developed to store this kind of data.

Graph database uses graph structure with nodes, edges and properties to represent and store data. Graph database is Index-free adjacency storage system. This means that every element contains direct pointer to every adjacent element and no

index look-ups are necessary. These are suited for the applications in which there are more interconnection between the data like social networks and web-site link structure, as the graph is natural way to represent relationship between user/nodes. Most common graph databases are Neo4j [44], OrientDB[45], AllegroGraph RDFstore [46] and HyperGraphDB [47].

Figure 2.7 represent schema of graph databases, here nodes represent entities like people, item. Properties are the attributes of these nodes and edges are the lines that connect nodes to each other. The edges represent relationships between nodes and can also have properties.

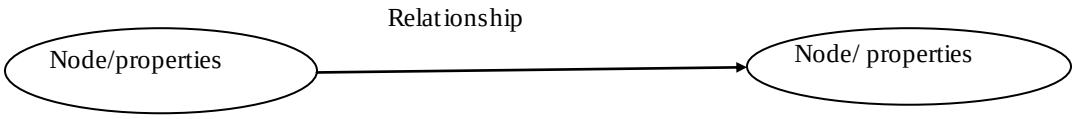

Figure 2.7: Representation of Graph in Graph Database

Figure 2.8 shows the example of graph database. In this example data is represented as graph. Attributes of the objects like Name, Id are stored as vertices and edges are used to represent the relationship between these objects like vedaang mittal is friend of veni goyal and he placed order with id 151.

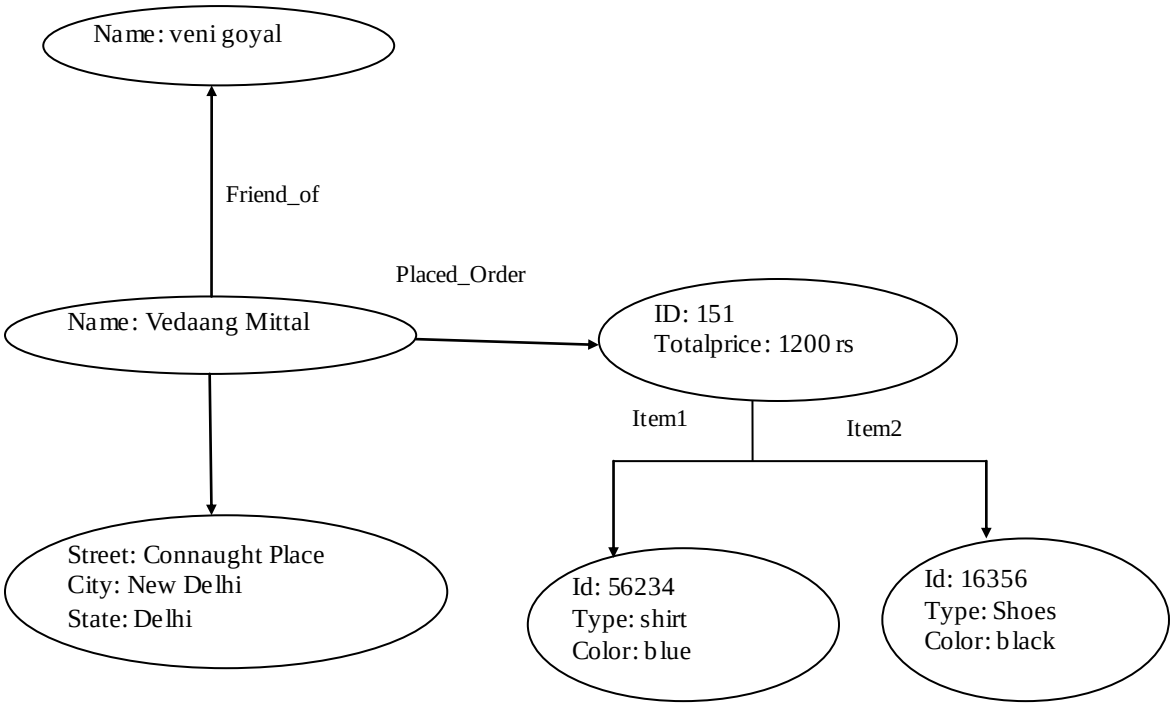

Figure 2.8: Example Graph

2.4.1 Neo4j

Neo4j is an open-source NoSQL graph database [44]. It is an embedded, disk-based, fully transactional java persistence engine that stores data structured in graphs rather than in tables [44]. It is implemented in Java, but it has binding for many languages like Python, Ruby, Jython, Scala. Neo4j is developed by Neo Technologies and has both AGPL and commercial licenses. It supports ACID transactions and master-slave replication. It is highly and mainly used in embedded applications. Cypher and Gremlin query languages can be used for graph traversals.

2.4.2 OrientDB

OrientDB is an open Source NoSQL database with both the features of Document and Graph databases [45]. Even if it is a document-based database, the relationships are managed as in graph databases with direct connections between records [45]. It supports natively HTTP, RESTful protocol and JSON without the use of 3rd party components. It uses MVRB-Tree as its indexing algorithm which is derived from the Red-Black Tree and B^{+} Tree and has fast insertion and ultra fast lookup. It supports ACID transactions and SQL language with extensions to handle relationships without Joins. It is purely written in java. It can work in schema-less mode, schema-full or a mix of both.

2.4.3 AllegroGraph RDFstore

AllegroGraph RDFstore is a persistent RDF graph database that purportedly scales to "billions of triples while maintaining superior performance" [46]. It is intended for building semantic web applications. It can store data and meta-data as RDF triples, which is standard format for linked data. AllegroGraph supports SPARQL, RDFS++ and prolog reasoning from various client applications. It is developed by Franz Inc. It supports ACID transactions (Commit, Rollback, Check pointing).

2.4.4 HyperGraphDB

HyperGraphDB is general purpose, open-source data storage mechanism based on a powerful knowledge management formalism known as directed hypergraphs [48]. This model allows a natural representation of higher-order relations, and is particularly useful for modeling data of areas like knowledge representation, artificial intelligence and bio-informatics [49].

Chapter 3

Problem Statement and Motivation

With the advent of Web 2.0 applications with millions of users reads and writes, a more scalable solution is required. The data stores for these applications needs to provide good horizontal scalability. Relational databases cannot solve these problems as they are not horizontal scalable. Similar issues are involved with cloud services, social networks, mobile usage and social media. Due to cloud services data is growing fastly and the data accumulated by social networks is more connected and semi-structured. So, there is need of the database that can store and process big data effectively and can satisfy the demand for high performance while reading and writing. Relational databases have many problems to cope with these trends. Technologies such as NoSQL were developed to meet the reliability and scalability needs. A growing number of developers and users have begun to use NoSQL databases. With this usage, there arose need to transform the existing applications from relational databases to NoSQL databases so that they can run on such platform.

Document-oriented databases are one of the categories of NoSQL databases that are appropriate for web-applications which involves storage of semi-structured data and dynamic queries. MongoDB document databases are able to face these new challenges as they allow horizontal scalability, support high-availability and have the flexibility to handle semi-structured data. MongoDB has typical applications in content management systems, mobiles, gaming and achieving.

Most of the transformation systems that have been developed till now, transform the data from relational to other non-relational databases (object-oriented databases and XML databases). However, to the best of our knowledge, no work has been done to transform a relational database into NoSQL databases. In this thesis a novel approach has been proposed for the transformation of relational database into document-based class of NoSQL database.

Chapter 4

Document-Oriented Databases

A document-oriented database is designed for storing, retrieving and managing document-oriented or semi-structured data. Like other NoSQL solution document database does not give ACID guarantees. Document-oriented database stores data in the form of documents unlike relational database which stores data in the form of normalized tables. The documents can be stored in XML, JSON, YAML or BSON data formats. In Figure 4.1 data stored in JSON based document database is illustrated. The document can be complex and the entire object model can be stored in one document and can be read and written at once. So, there is no need to create a series of insert statements and complex procedures. The documents are independent. It improves the performance and decreases side effects of concurrency. Most document databases support versioning of documents with the flip of switch.

Documents are stored in collections. Compared to relational database, document corresponds to record (tuple) and collection corresponds to table. But like relational table collection does not enforce fixed schema. A collection can store documents with completely different set of attributes and new attributes can be added to a document without affecting another document in a collection.

"Person":	

{
"First Name" : "Anjali"
"Last Name" : "Kanna"
"Likes" : ["books" , "movies"]
"age" : 20
}
	{
	"First Name" : "Rohit"
	"Last Name" : "Verma"
	"Likes" : ["Cricket"]
	}

Figure 4.1: Document Data Store in JSON Format

Documents can be mapped directly to the class structure of programming language but it is difficult to map RDBMS entity relationship data model. This makes easier to programming with document database. Due to arrays and embedded documents, there is no need of Joins in document databases. Two of the most popular document-oriented database systems are MongoDB and CouchDB.

4.1 Comparison of Document-Oriented Databases

Comparison of three popular document-oriented databases MongoDB, CouchDB and RavenDB is shown in Table 4.1.

Table 4.1- Comparison of Document-Oriented Databases

	MongoDB	**CouchDB**	**RavenDB**
Document data format	BSON	JSON	JSON
Written in	C++	Erlang	C#
Document Versioning	No	Yes	Included plug-in
Adhoc Query	Yes	No	No
Map/Reduce	JavaScript + others	JavaScript	LINQ
Sharding	Yes	Yes	Yes
Transactions	No	No	ACID
Replication	Master-slave Replica Sets	Master-slave	Master-slave replication Master-master replication
Concurrency	In-place Update	Multi Version Concurrency Control (MVCC)	Optimistic Concurrency
Consistency	Strong Master/ Eventual Slave	Strong Master/ Eventual Slave	Eventual
Security	Basic	Basic	Basic using included plug-in
Interface	Custom Protocol over TCP/IP	HTTP/REST	HTTP/REST
Indexing	B-Tree	B^+-tree	Lucene.Net

- RavenDB provides ACID guarantees when saving document through optimistic concurrency. CouchDB also guarantees ACID properties when saving documents through an MVCC mechanism. MongoDB does not support concurrency and updates documents in-place. This means MongoDB will be

much faster for writing and scale horizontally very easily at the expense of guaranteed data consistency [50].

- CouchDB and RavenDB stores data in JSON format, while MongoDB serialize JSON to BSON (Binary JSON). The main advantage of BSON over JSON is efficiency. So, the computations in MongoDB are faster and the data is smaller.
- One advantage of MongoDB is that the queries are simple. But in CouchDB there is need to write Map/Reduce for simple queries
- CouchDB does not have collections. So, it is one big data store. While MongoDB allows to group data into collections. This can make the query speed faster.

It can be seen that MongoDB is better than other document databases in above aspects. Due to these advantages, for our case study MongoDB is selected over other document-databases.

4.2 MongoDB

MongoDB is an open source NoSQL document database, initiated by 10gen Company. It is written in c++ and its query language is javascript. The word mongo in its name comes from word humongous [40]. MongoDB does not support Joins or ACID transactions. MongoDB was designed to handle growing data storage needs.

MongoDB provides interactive JavaScipt shell for database management. It supports a rich, ad-hoc query language of its own. In addition, there exist bindings for many programming languages like Java, C, C++, Erlang, Python, Ruby and more. The data type of many languages is built-up of key-value pairs and some languages support JSON. It is easy to create documents for MongoDB using these languages.

MongoDB keeps all of the most recently used data in RAM. When indexes are created for queries, all data sets fits in RAM and queries are run from memory. There is no query cache in MongoDB. All queries are run directly from the indexes or data files. Data is physically written to the disk with in 100 milliseconds.

The main purpose of MongoDB is to support mass data storage. To support massive data storage, it also comes with distributed file system GridFS. When the amount of data is more, access speed of MongoDB is 10 times more than the speed of MySQL. So, it solves the problem of problem of massive data access efficiency.

4.2.1 Data model

MongoDB stores data in the form of collections. Each collection contains documents. Due to document structure, it is schema less. It is easy to change the structure of model. MongoDB documents are stored in binary form of JSON called BSON format. BSON supports Boolean, float, string, integer, date and binary types. It is easy to traverse and parse the BSON documents. User enters the data into document in the JSON format, these data then converted into BSON format and stored. When user retrieves the data, the data again converted from BSON to JSON form.

MongoDB uses dynamic schema. There is no need to define the structure (fields and types of their values) before creating the collections. Structure of the documents can easily be changed by adding or removing fields and documents in a collection need not to have identical set of fields.

4.2.2 Integrity rules

Unlike relational databases that offer a wide range of integrity rules, MongoDB only can have uniqueness of documents in a collection. MongoDB automatically generates unique “id” field (object id) for documents. In addition to unique id, unique indexes can also be defined on attributes or combination of attributes. Since MongoDB has no built-in support for joining different documents, it does not offer any rules for referential integrity. Instead data from two or more documents can be stored together, known as embedding.

4.2.3 Indexing

Database indexes play an important role to optimize the “read” queries. In relational databases primary key is used as default index. Similarly automatically generated “id” field is used as default index in MongoDB. Any attribute or combination of attribute of a document can be indexed using compound indexing. All indexes are implemented as B-trees. The ensureIndex() function is used to create indexes from JavaScript shell.

4.2.4 Sharding and Replication

MongoDB has inherent support for replication. Replication means database is synchronized on two or more computers. MongoDB uses asynchronous replication for redundancy and failover. It supports two types of replication.

- **Master-slave replication:** In this replication master is able to access and write and slaves are only replica sets (used for back-ups).
- **Replica-set:** In this group of nodes work together to provide automated failover. In this replication another master can be elected in case of failover.

Another important feature of MongoDB is "sharding". In sharding collections are distributed over multiple nodes. All documents of a collection should have shard key. The shard key is used to distribute the collections. MongoDB also have "load balancing" feature. When nodes contains different amount of data, MongoDB automatically redistribute the data so that load is equally distributed across the nodes.

4.3 MongoDB Production Users

MongoDB is general purpose tool and can be used for different projects. Many organizations are using MongoDB for archiving, Education, web, mobile, Ecommerce, content management, cloud, social networks and other enterprise solutions. A few of the most popular production users are as follows [51].

- **Craigslist**: In today environment, the volume of data to manage is increasing and requirements are changing. Craiglist is popular classifieds and job posting community that serves 570 cities in 50 countries. It needs to archive years of accumulated data, where structure of data has changed numerous times. With 1.5 million new classified ads posted every day, Craigslist must archive billions of records in many different formats, and must be able to query and report on these archives at runtime. Historically, Craigslist used MySQL cluster to store the information. But in MySQL simple schema change on their vast archive took months to complete, preventing them from pushing new features. Craigslist also faced the problems of flexibility and cost management with MySQL. Due to scalability and flexible schema of MongoDB, they found MongoDB more suitable for their needs. In 2011, Craigslist migrated, over two billion documents to MongoDB from their archive of classified ads.
- **MTV Networks**: MTV Networks owns and operates hundreds of high-traffic web properties, including spike.com, gametrailers.com, thedailyshow.com, comedycentral.com, and nick.com. Due to document data model and flexible schema of MongoDB, MTV uses it as the data store for a new unified Java-based content management system (CMS).

- **SAP**: MongoDB is used as a core component for the SAP's platform-as-a-service (PaaS) offering. It is used for the Enterprise Content Management (ECM). Due to flexibility and scalability of MongoDB, SAP can scale its content management service on its PaaS offering to meet customer requirement.
- **Disney:** Disney Interactive Media Group (DIMG), the interactive entertainment affiliate of The Walt Disney Company, creates immersive, connected, interactive experiences used by millions of people every day across console gaming, online, mobile, and social networking platforms. Because of the diversity of interactive media platforms and the number of operating groups, almost every game and content site managed its data storage independently. Looking to reduce time-to-market, DIMG developed a common platform for data storage and processing. Using MongoDB, Disney was able to build a scalable platform that all gaming divisions could leverage, supported by a single, centralized team.
- **The Guardian:** Guardian.co.uk is a leading UK-based news website. They have spent ten years fighting for relational database representation of domain model. In 2011, they started using MongoDB, taking advantages of flexible documents and query flexibility.
- **Forbes:** Forbes is nearly 100 years old and is the largest U.S-based business media brand. An online version of the print publication was established more than a decade ago, with 2% of its content originating from the magazine and the remaining 98% exclusively created for the web. The changing media landscape and the drive for more dynamic content required a nimble site that didn't involve reengineering with every addition or change. Now, Forbes.com stores all of its core assets, including data for articles and the company's renowned Lists (e.g. World's Richest People, Best Places to Live), in MongoDB. As Forbes.com's content serving platform, MongoDB delivers ease of use, flexibility and TCO and has transformed how Forbes.com's data is presented to the web consumer.
- **SourceForge:** SourceForge is a web-based source code repository. It acts as a centralized location for software developers to control and manage free and open source software development. MongoDB is used for back-end storage on

the SourceForge front pages, project pages, and download pages for all projects.

- **Intuit:** Intuit is one of the world's largest software and service provider for small businesses. For small businesses websites network, Intuit uses MongoDB to track user engagement and activities in real time.
- **GitHub:** GitHub is a web-based hosting service for software development projects that use the Git revision control system. GitHub offers both paid plans for private repositories, and free accounts for open source projects. GitHub is most popular social coding site. It uses MongoDB for an internal repository application.
- **Springer:** Realtime.Springer.com is a service that accumulates downloads of Springer journal book chapters and articles in real time and display these download in different views. The main purpose of this service is to provide information of journal daily usage of the scientific community. MongoDB is used to store the details of downloads per day from Springer's sites
- **Viber:** Viber media offers free phone calls and text messaging between users over 3G and wireless networks in iPhone and Android applications. Viber is using MongoDB as core cloud infrastructure due to scaling ability as more users can join viber community and there is need to manage the unpredictable data loads from its millions of mobile users.
- **Foursquare:** Foursquare is a location-based social network that allows users to "check-in" to venues on their mobile phones to earn points and rewards. Growing rapidly since its inception in 2009, the company needed to efficiently scale their application with limited engineering resources. As their data grew, foursquare made the strategic decision to migrate storage of venues and check-ins from their original relational architecture to MongoDB.
- **Wordnik:** Wordnik is the world's largest English language resource. It is an online dictionary that is six times bigger than the Oxford English Dictionary. Wordnik uses MongoDB as the foundation for its live dictionary. MongoDB offers the high performance and reliability the company required, and today, Wordnik stores its entire text corpus in MongoDB - 3.5T of data in 20 billion records.

- **Bit.ly:** Bit.ly is a bookmarking service and URL shortening owned by Bitly, Inc. It allows users to shorten, share, and track links. Bit.ly uses MongoDB to store user history
- **OpenSky:** OpenSky is a free online platform that helps people to discover amazing products. People can share these products with their friends and family and can earn money. MongoDB is used by OpenSky for e-commerce.
- **ShopWiki:** ShopWiki uses MongoDB as a data store for its shopping search engine that brings many stores and products for online shopping needs of users. Due to flexibility offered by MongoDB, ShopWiki is using it as a storage engine for all R&D and data mining efforts. ShopWiki uses MongoDB in cases where the usage of MySQL would not be practical,
- **The New York Times:** The New York Times (or NYT) is an American daily newspaper, founded and continuously published in New York City since September 18, 1851. Its website is America's most popular news site, receiving more than 30 million unique visitors per month. The New York Times uses MongoDB for photo submissions in a form-building application. Because of MongoDB's dynamic schema, producers can define any combination of custom form fields.
- **Sailthru:** Sailthru is an innovative email service provider that focuses on improving the quality of emails over quantity. Due to flexibility and schema-free data storage Sailthru uses MongoDB. As MongoDB use arbitrary JSON format, customers can use objects and arrays inside of their emails.

Chapter 5

Design and Implementation

Relational databases have been used for data management for decades. But NoSQL databases have gained popularity in the recent years. Now a growing number of organizations and users are using NoSQL databases. They are migrating from relational databases to NoSQL databases. So, there is need to transform the data from relational databases to NoSQL databases. It also needs to be known how data is modeled and queried in NoSQL databases. In the following sections, modeling and querying in NoSQL database is shown. MongoDB document database is chosen as NoSQL database. A transformation algorithm is also proposed. The proposed algorithm is implemented by using NetBeans IDE. For this purpose, MySQL is selected as relational database and MongoDB as document-oriented NoSQL database.

5.1 MySQL

MySQL [52] is most popular open source relational database management system. It is owned by Oracle. It is used in a variety of projects and is stable. MySQL is fast, robust, easy to use, multi-user and multi-threaded SQL database server. MySQL support ACID transactions and foreign keys. Because of its simple installation and setup procedure, it is used by small companies as well as large production environments such as Twitter. In Twitter, it is used in the way of a key-value store [13]. MySQL is written in C and C++. It has drivers available in most programming languages that allow programs to access the API. An interactive shell is provided for SQL queries. The programs that use the provided native drivers can also use SQL to interact with the database.

The relational schema of database used as example for implementation and design is shown in Figure 5.1. The database is based on Post-Comment system. A user can post any number of posts at any time on the site. User can also give comments and sub-comments under any post. Tags are used to classify the posts. The database consists of four tables: User, Post, Comment and Tag. User table attributes are uid, username, realname, email, homeurl, pswd. Uid is primary key. The user can login into the system by using username and password. So, user must have valid account. Homeurl is the url of its home page. Post table has pid, uid, title, body,

primaryurl, and time attributes, where pid is the primary key of the post table. Uid is foreign key that refers to the primary key of the user table. Any user can post an article without having valid uid. For an outsider, uid is null. Attributes of comment table are cid, parented, postid, userid, title, comment, time, score, descriptor, where cid is the primary key of table. Nested comments are also allowed. Nested comments are comment under a comment. Postid is the id of the post under which user has commented. Parentid is the id of the comment under which sub-comment is written. Attributes of tag table are tagid, name, pid. Tagid is primary key. Pid is foreign key which refers to the pid of the post table. Tags are used to categorize the posts.

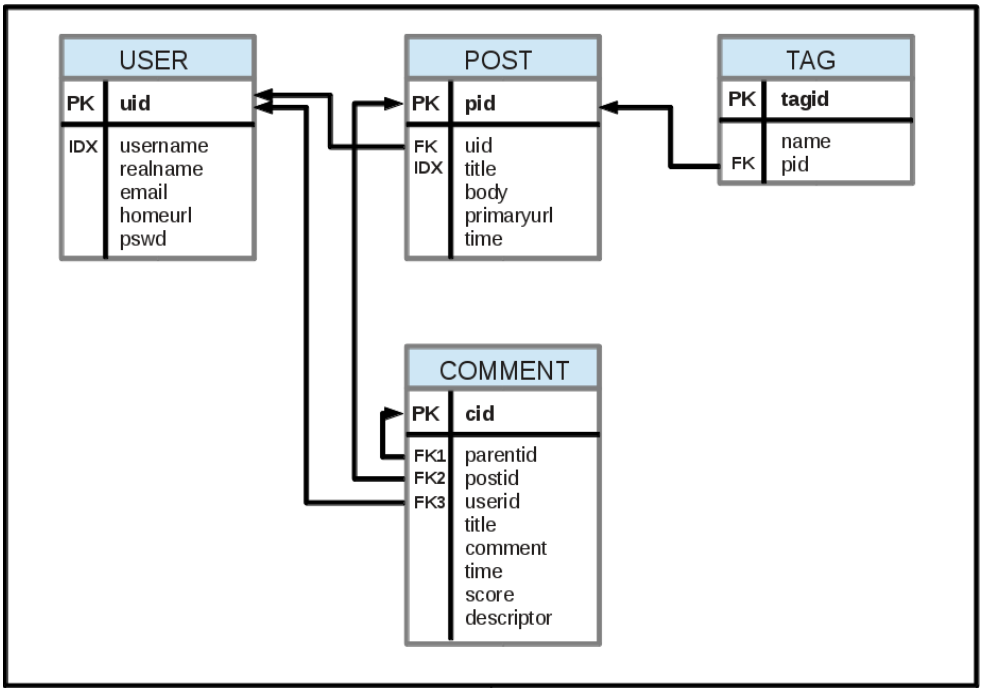

Figure 5.1: Schema of Source Relational Database

5.2 MongoDB

MongoDB is an open source document-oriented NoSQL database [40]. MongoDB stores data in the form of collections. Collection contains documents. Documents are stored in the binary JSON format (BSON). Compared to relational database, collection corresponds to table and document corresponds to relational tuple. However, unlike relational database, collection in MongoDB do not enforce fixed schema. Documents in the same collection do not need to have the same set of fields or structure. Common fields in a collection's documents may hold different types of data.

MongoDB does not use Joins to relate documents like Relational database. In MongoDB data is stored in the denormalized form in which related data is stored in

single document. This is known as Embedding. Embedding provides good performance for read-intensive applications, as related data can be retrieved in single database operation. The problems with fully embedded structure occur while writing documents to the collection. The need to store the data in different documents arises when data starts to duplicate during embedding. To represent many-to-many relationships data is stored in different documents. To indicate the relationships between the data represented by documents, a reference is stored between two documents. Referencing provides more flexibility than embedding.

The database used as example, which has to be transformed from MySQL to MongoDB contains three collections: User, Tag and Post. Post contains an embedded document named comments. Class diagram and JSON format is used for modeling schema of the database.

5.2.1 Class diagram representation

Class diagram can be used to represent the schema in MongoDB. Documents are represented by classes. Embedded documents and referencing between documents can be represented by relationships between the classes. Composition is used to represent the embedded documents and associations are used to represent the referencing. Document fields are represented by class attributes. Figure 5.2 shows class diagram representation of database used as example. Post and Comment are embedded documents represented by composition. Comment exists only if the post corresponding to that comment exists that is comment is embedded into the post. Post, User and Tag are different documents. To relate these documents reference is used. In post document, uid is used as reference field of user document and tagid is used as reference of tag document to indicate a relationship between documents. In user document pid is used as reference of the post document and cid is used as reference of comment document. In comment document userid is used as reference of the user document and in tag document pid is used as reference of the post document.

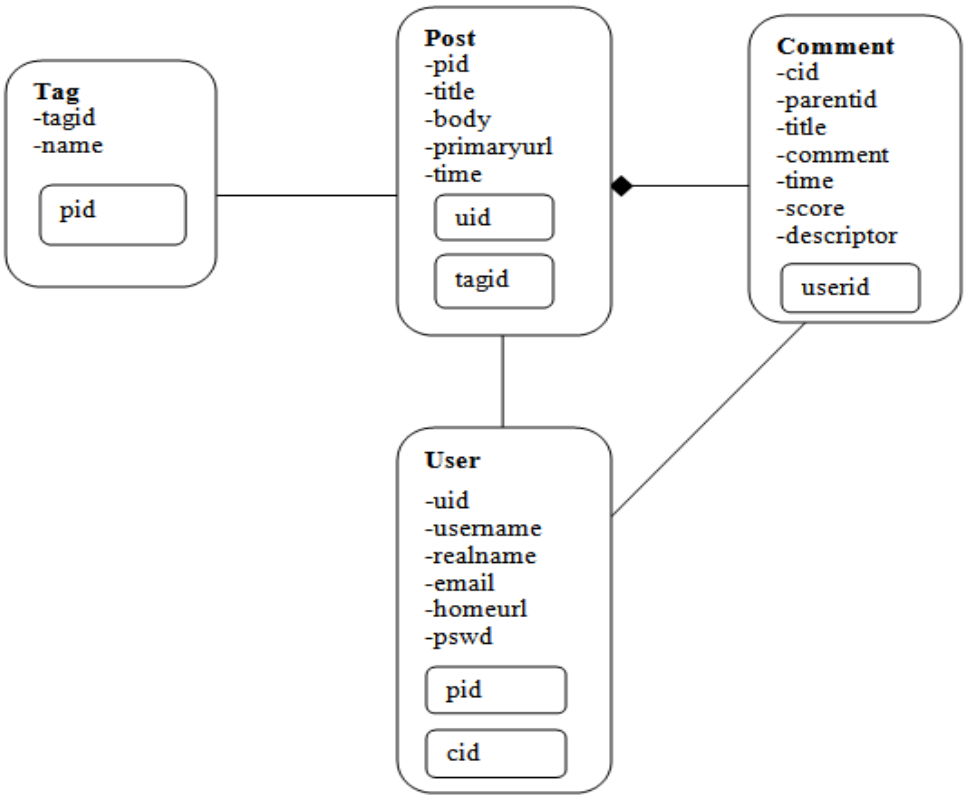

Figure 5.2: Class Diagram Representation of Sample Database

5.2.2 JSON format representation

JSON (JavaScript Object Notation) is a lightweight and highly portable data-interchange format. It is easy for humans to read and write JSON documents. It is originated from JavaScript scripting languages and is represented with two primary data structures. One is ordered which is known as array. Other is name/value pair, known as object. JSON is language independent with parser available for many languages. JSON is inherent to the web as well as browser and often used for serializing and transmitting structured data over a network connection.

In MongoDB documents, data is represented in the form of binary JSON format that is in the form of field and value pairs. A field-value pair comprises of a "field name" in double quotes, followed by colon ":" and then "value" in double quotes. The value can be another documents, arrays and array of documents. Each pair is separated by comma. Documents are held within curly ("{ }") brackets and arrays are held within square ("[ ]") brackets. Document structure of database used for transformation is shown in Figure 5.3. Post, User and Tag are collections. "_id" field is unique-id field and may contain value of any BSON data type other than array. In post collection, tagid is array of strings and comments is array of sub documents.

```
"Post":
{
        "_id":"<string val>",
        "uid":"<integer val>",
        "title":"<string val>",
        "body":"<string val>",
        "primary url":"<string val>",
        "time":"<datetime val>",
        "tagid":["<string val>"],
         "comments":[
              {
                    "cid":"<string val>",
                     "parented":"<string val>",
                    "userid":"<integer val>",
                    "title":"<string val>",
                    "comment":"<string val>",
                     "time":"<datetime val>",
                      "score":"<number val>",
                      "descriptor":"<string val>"
              }
        ]
}
```

```
"User":
    {
        "_id" :"< integer val>",
        "username": "<string val>",
        "realname": "<string val>",
        "email":"<string val>",
        "homeurl":"<string val>",
         "pswd":"<string val>",
         "pid":["<string val>"],
         "cid": ["<string val>"]
    }

"Tag":
    {
        "_id": "<string val>",
         "name":"<string val>",
         "pid":["<string val>"]
    }
```

Figure 5.3: JSON Format Representation of Sample Database

5.3 Querying MySQL and MongoDB

This section describes how queries are written in MongoDB. Seven Queries are designed to describe syntax in MongoDB. The syntax of same queries is also explained in MySQL.

Query 1: Find the tag names which have been used in the post under which a particular user has commented. (user='a1').

MySQL	mysql> select t.name from comment c,post p,user u,tag t where p.pid=c.postid and c.userid=u.uid and p.pid=t.pid and u.username='a1'; **Output:** +--------+ | name | +--------+ | space | | apple | | google | | apple |

	+--------+
MongoDB	> var u=db.user.findOne({username:"a1"}) >var tag1=db.post.find({"comments.userid":u._id}) > db.tag.find({_id:tag1.tagid},{_id:0,name:1}) **Output:** { "name" : "space" } { "name" : "google" } { "name" : "apple" }

The find () method selects documents from a collection that meets the <query> argument. <Projection> argument can also be passed to select the fields to be included in result set. The find () method returns a cursor to the results. This cursor can be assigned to variable.

db.collection.find(<query>,<projection>)

The findOne() method is similar to find() method but it selects only one document from a collection.

In this query first the document with username 'a1' is extracted from user collection and stored in variable u. Then the comments corresponding to that user are found by matching u._id field with userid field of subdocument comments (comments.userid) in Post Collection and returned cursor is stored in the variable tag1. Then _id is matched in Tag collection. To exclude _id field, {_id:0} is written in projection field and the fields which need to be included are written as {fieldName: 1}. For instance, in above query, name to be included is written as {name: 1} in projection argument.

Query 2: Find the tag(s) which have been used in the posts of each user.

MySQL	mysql>select u.uid,t.tagid from user u,post p,tag t where u.uid=p.uid and p.pid=t.pid; **Output:** +-----+-------+ | uid | tagid | +-----+-------+ | 101 | tag1 |

	| 101 | tag3 | | 102 | tag2 | | 102 | tag2 | | 102 | tag3 | +-----+-------+ 5 rows in set (0.09 sec)
MongoDB	db.post.find({},{_id:0,uid:1,tagid:1}) **Output:** { "uid" : 101, "tagid" : ["tag1", "tag3"] } { "uid" : 102, "tagid" : ["tag2"] } { "uid" : 102, "tagid" : ["tag2", "tag3"] }

Query 3: Given a cid of a comment find its related post, parent-comment and tags associated with the post.

MySQL	mysql> select c.cid, c.postid, c.parentid, t.tagid from comment c,post p ,tag t where c.postid=p.pid and p.pid=t.pid and c.cid='c2'; **Output:** +-----+--------+----------+-------+ | cid | postid | parentid | tagid | +-----+--------+----------+-------+ | c2 | p1 | c1 | tag1 | | c2 | p1 | c1 | tag3 | +-----+--------+----------+-------+ 2 rows in set (0.05 sec)
MongoDB	db.post.find({"comments.cid":"c2"},{"comments.cid":1,"comments.parentid":1,tagid:1}); **Output:** { "_id" : "p1", "tagid" : ["tag1", "tag3"], "comments" : [{ "cid" : "c1" }, { "cid" : "c2", "parentid" : "c1" }] }

Query 4: Find the time of the post under which some user has commented. (user='a1').

MySQL	mysql> select p.time from comment c,post p,user u where

	p.pid=c.postid and c.userid=u.uid and u.username='a1'; **Output:** +--------------------+ | time | +--------------------+ | 2012-10-13 03:20:00 | | 2012-10-23 23:55:00 | +--------------------+ 2 rows in set (0.00 sec)
MongoDB	> var u=db.user.findOne({username:"a1"}) > db.post.find({uid:u._id},{time:1,_id:0}) **Output:** { "time" : ISODate("2012-10-13T03:20:00Z") } { "time" : ISODate("2012-10-23T23:55:00Z") }

Query 5: List the posts of each user.

MySQL	mysql> select u.uid,p.pid from user u,post p where u.uid=p.uid; **Output:** +-----+-----+ | uid | pid | +-----+-----+ | 101 | p1 | | 102 | p2 | | 102 | p3 | +-----+-----+ 3 rows in set (0.22 sec)
MongoDB	db.post.find({},{_id:1,uid:1}) **Output:** { "_id" : "p1", "uid" : 101 } { "_id" : "p2", "uid" : 102 } { "_id" : "p3", "uid" : 102 }

Query 6: Find the total number of posts by a particular user. (uid =102)

MySQL	mysql> select count(p.pid) from post p,user u where u.uid=p.uid and u.uid=102; **Output:** +--------------+ | count(p.pid) | +--------------+ | 2 | +--------------+ 1 row in set (0.00 sec)
MongoDB	> db.post.count({uid:102}) **Output:** 2

Query7: Find out the username, uid, tags associated with posts of a comment and score associated with comments.

MySQL	mysql> select c.cid, u.username,c.userid,c.score, t.tagid from comment c,user u,tag t where c.userid=u.uid and c.postid=t.pid; **Output:** +----------+--------+-------+-------+ | username | userid | score | tagid | +----------+--------+-------+-------+ | a1 | 102 | 2 | tag1 | | a1 | 102 | 2 | tag3 | | a0 | 101 | 1 | tag1 | | a0 | 101 | 1 | tag3 | | a0 | 101 | 3 | tag2 | | a0 | 101 | 3 | tag3 | | a1 | 102 | 2 | tag2 | | a1 | 102 | 2 | tag3 | | a0 | 101 | 4 | tag2 | | a0 | 101 | 4 | tag3 | +----------+--------+-------+-------+

	10 rows in set (0.00 sec)
MongoDB	>db.post.find({},{tagid:1,_id:0,"comments.score":1,"comments.userid":1,"comments.cid":1}) **Output:** { "tagid" : ["tag1", "tag3"], "comments" : [{ "cid" : "c1", "userid" : 102, "score" : 2 }, { "cid" : "c2", "userid" : 101, "score" : 1 }] } { "tagid" : ["tag2"] } { "tagid" : ["tag2", "tag3"], "comments" : [{ "cid" : "c3", "userid" : 101, "score" : 3 }, { "cid" : "c4", "userid" : 102, "score" : 2 }, { "cid" : "c5", "userid" : 101, "score" : 4 }]

5.4 Pentaho data integration

The massive changes in business intelligence have changed the way the industry and customers thinks about analytics. The exponential growth in big data, driven by the broad proliferation of data, is a key driver of this change. In addition, the growth of big data is accelerated by cloud services like Google, Amazon. With growing volumes and varieties of data flowing at increasing speed, organizations need a fast and easy way to harness and gain insight from their big data sources. Due to business analytics requirements demanded by customers, there arose need of new tools that can collect all types of data, and to store, manage, manipulate, aggregate, analyze and integrate all that data into useful ways that positively impact their business.

Pentaho is one of the business analytic tools. Pentaho brings together IT and business users to easily access, explore and analyze all data that impacts business results. Pentaho and 10gen also offers MongoDB-based big data analytics solution to the market. This solution combines MongoDB with Pentaho's visual interfaces for high-performance data input, output and manipulation, as well as data discovery, visualization and predictive analytics. This makes it easy and productive for IT staff, developers, data scientists and business analysts to operationalize, integrate and analyze both big data and traditional data sources [53]. Benefits of Pentaho for MongoDB are shown in Figure 5.4.

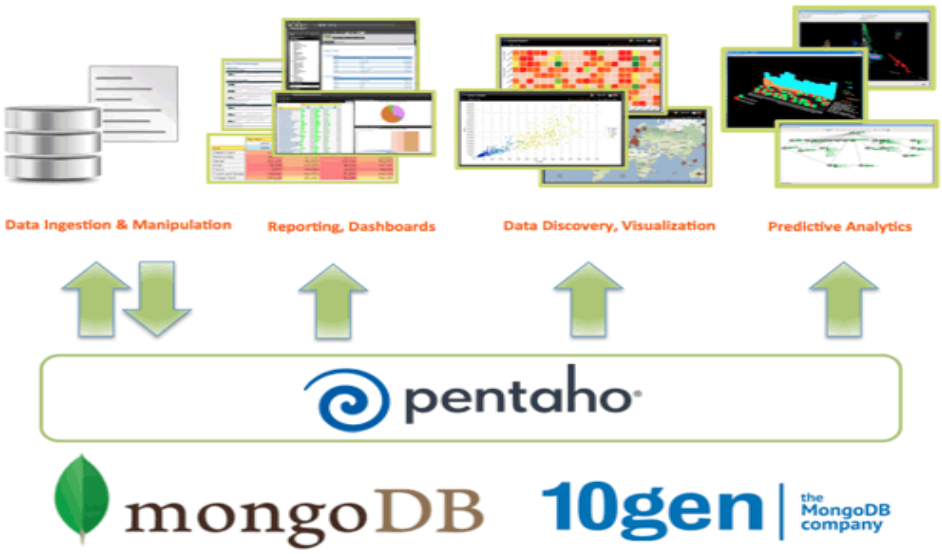

Figure 5.4: Benefits of Pentaho for MongoDB [53]

5.5 Proposed Algorithm

Based on the data model and features of the MongoDB, an algorithm has been proposed for data transformation from relational database to MongoDB document-oriented database.

Algorithm: DataTransformation

Input: Source database as S
Output: Target database as T

```
1.  String tbl[]:= get_table_names(S)
2.  String embdtbl[]:= get_embed_table_names(tbl[])
3.  String embedded:= get_embedded_table(embdtbl[])
4.  for all i € {1. . tbl.len} do
5.      integer toBeEmbedded:=0
6.      for all j € {1. . embdtbl.len} do
7.          if (tbl[i]==embdtbl[j]) then
8.              toBeEmbedded:=1
9.      end for
10.     if (! toBeEmbedded)
11.         String colnames[]:=get_column_names(tbl[i])
12.             for all k € {1. . colnames.len} do
13.                 String colvalues[]:= get_column_values(colnames[k])
14.             end for
15.         generate_tbl[i]_text_file (colnames[] , colvalues[])
16.         String PK[]:= get_primary_keys(tbl[i])
17.         String ETN[]:= get_exported_keys_table_names(tbl[i])
18.         Boolean ETNcheck:= check_for_null(ETN[])
```

```
19.                 if(!ETNcheck) then
20.                     String EPK[] := get_exported_tables_primary_keys (ETN[])
21.                     Generate_tbl[i]_keys_text_file (PK[], EPK[])
22.                 end if
23.                 else
24.                     Generate_tbl[i]_keys_text_file (pk[])
25.       end if
26.   end for
27.   String clmJoin[]:= get_col_for_join(embdtbl[])
28.       for all m € {1. . embdtbl.len} do
29.           if (embdtbl[m] == embedded) then
30.               String embeddedtbl:= embdtbl[m]
31.           else
32.               String embeddingtbl:= embdtbl[m]
33.       end for
34.   String      colnames[]:=get_column_names(embeddedtbl      LEFT      JOIN
      embeddingtbl)
35.       for all k € {1. . colnames.len} do
36.           String colvalues[]:= get_column_values(colnames[k])
37.       end for
38.   generate_embedding_text_file (colnames[],colvalues[])
39.   String PK[]:= get_primary_keys(embeddedtbl])
40.   String ETN[]:= get_exported_keys_table_names(embeddedtbl])
41.   boolean ETNcheck:= check_for_null(ETN[])
42.       if(!ETNcheck) then
43.           String EPK[] := get_exported_tables_primary_keys (ETN[])
44.           generate_embeddedtbl_keys_text_file (PK[], EPK[])
45.       end if
46.       else
47.           generate_embeddedtbl_keys_text_file (pk[])
48.   Intergrate_with_pentaho ( )
49.   Load_data_into_MongoDB( )
```

5.6 Implementation

The proposed algorithm has been implemented by using NetBeans Java IDE. For this purpose, MySQL is chosen as relational database. Data is transformed from relational database (MySQL) to document database (MongoDB). The steps involved in the transformation are explained below.

Step1: In first step, connection with the relational database (i.e. MySQL) is established. Once the successful connection has been made to the MySQL server, then all the existing databases from relational database are extracted and shown into the combo box as shown in Figure 5.5.

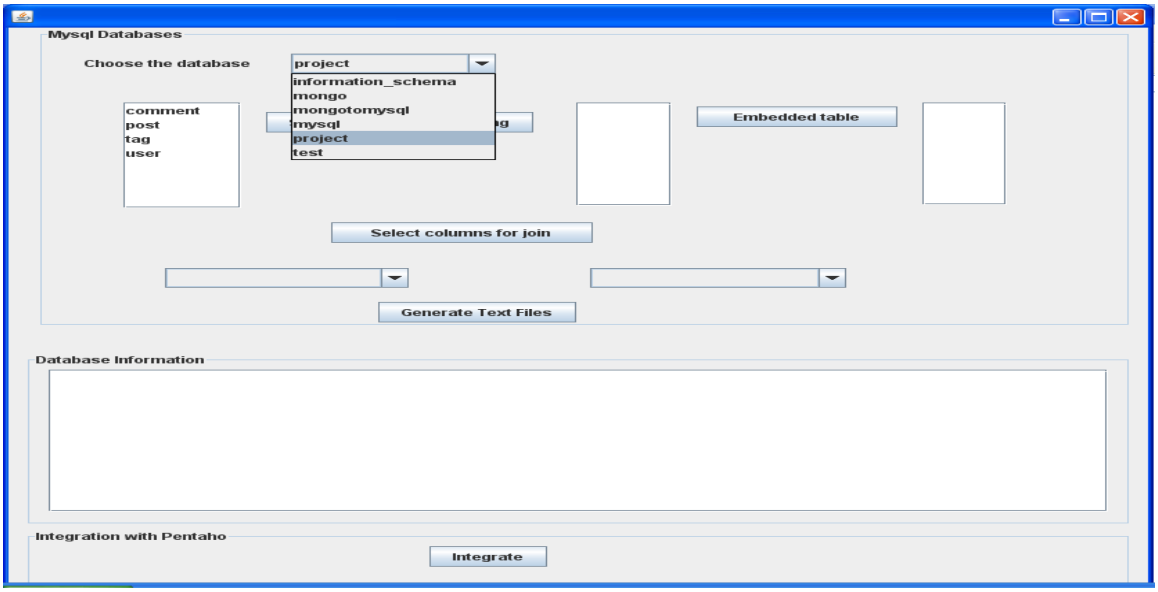

Figure 5.5: Choose the Source Database to be transformed

Then user chooses the database which needs to be created in MongoDB. That is the database, whose data is to be transformed into MongoDB is chosen. Here, project is chosen from existing databases for transformation as shown in Figure 5.5. All the tables from the chosen databases are shown into the list. The chosen database project has four tables: comment, post, tag, and user.

Step 2: From the table list extracted in step 1, the tables whose corresponding embedded documents are to be created in MongoDB, are chosen. In MongoDB, related data of two or more tables can be stored in a single document. This is known as embedding and documents are known as embedded documents.

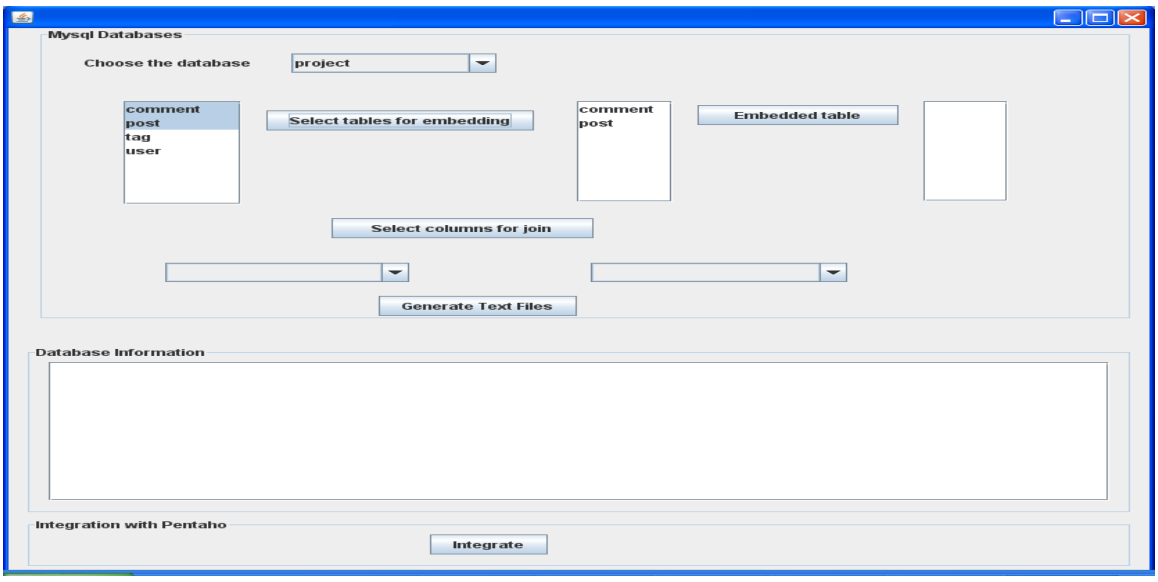

Figure 5.6: Table Selection for Embedding

As shown in Figure 5.6, comment and post are chosen as embedded document. To choose comment and post, the tables are first selected and then ‘select tables for embedding’ button is clicked. The tables are displayed into the second list.

Step 3: As can be seen in Figure 5.7, comment and post are shown into the second list. From these two documents choose the one in which other document has to be embedded. Here post is chosen as embedded document, it means post contains comment. To choose the document, first the document is selected and then button ‘Embedded table’ is clicked and the table name is shown in the list.

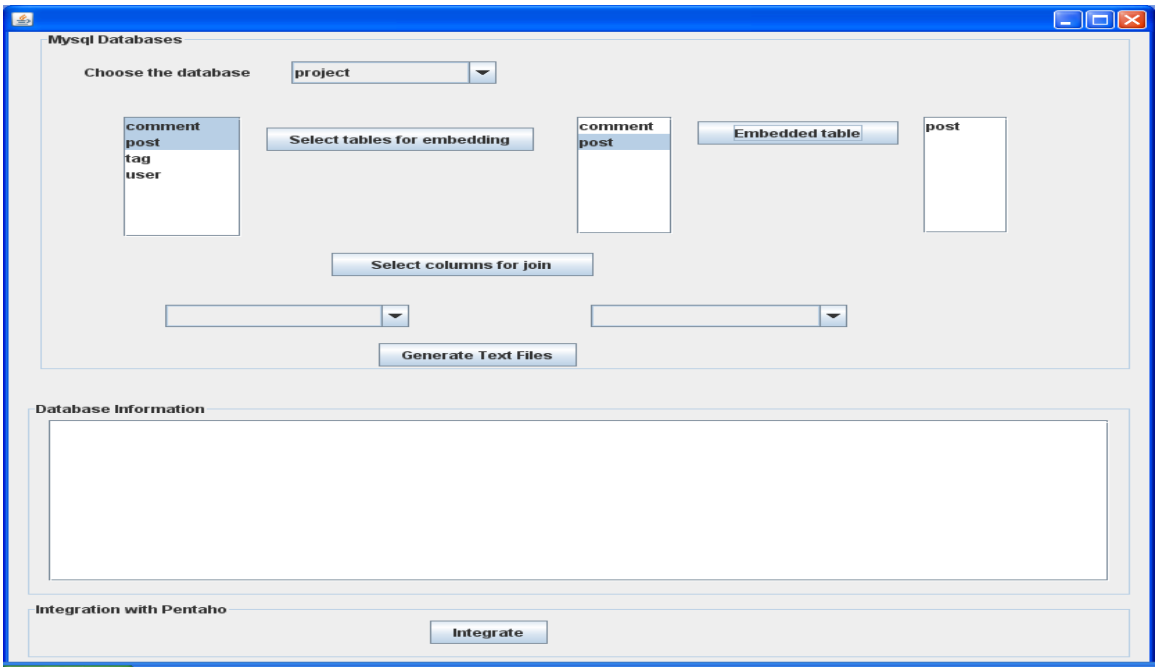

Figure 5.7: Selection of Embedded Table

Step 4: In this step, the columns of embedded table (post and comment, which are chosen in second step) on which Join has to be applied are chosen. Joining of tables is required to generate a single embedded document corresponding to two tables. For this purpose, first ‘Select columns for joins’ button is clicked and then the columns of both the selected embedded tables are shown into the combo box. Columns for Joins are chosen from these columns. Here pid column of both the tables is chosen for Join as shown in Figure 5.8.

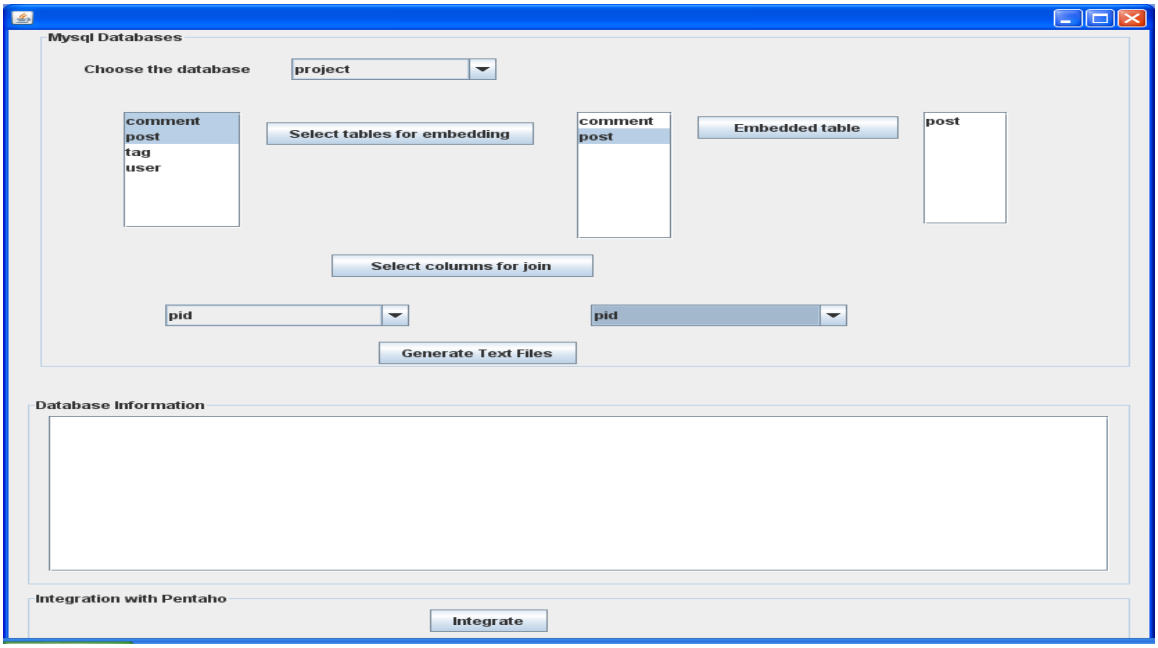

Figure 5.8: Selection of Columns to Join Tables

Step 5: Pentaho takes the text files (in a specific format) as input to write the data into the MongoDB. This file should contain column names and column values. So, first these files need to be generated. To generate the text files 'Generate Text File' button is clicked. The text files corresponding to tables are generated separately but only one file is generated corresponding to embedded documents. The information of all the generated files is shown into text area as shown in Figure 5.9.

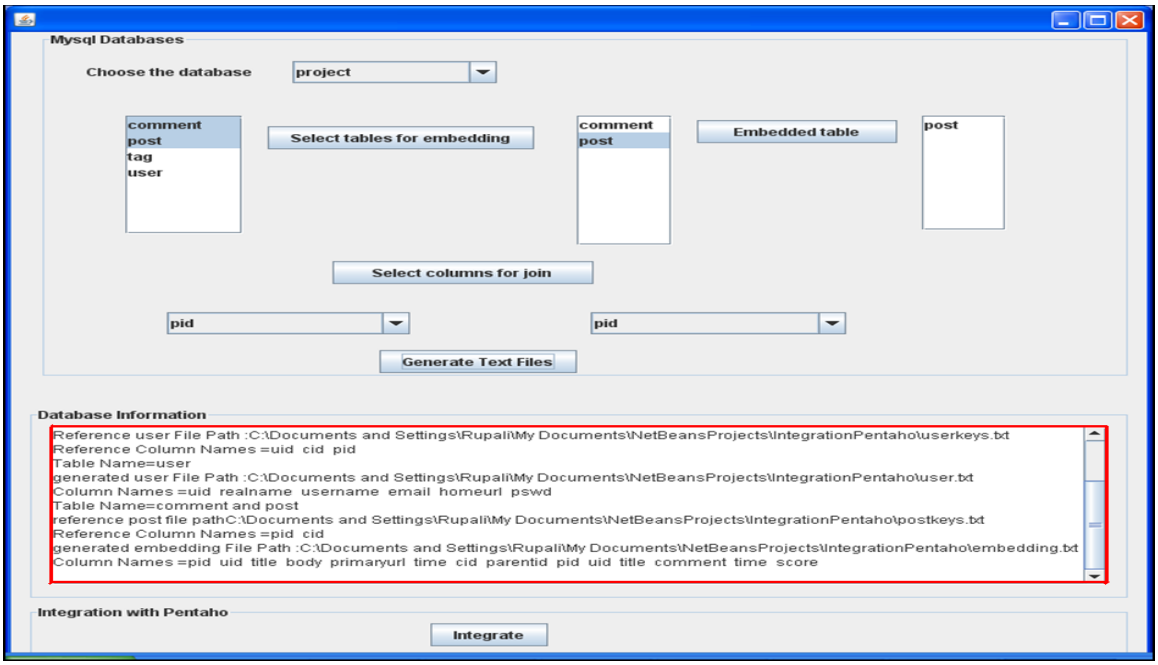

Figure 5.9: Text Files Generation

Here, table name is the name of the table, whose files (reference file and text file) are generated. Reference file path is the path of file used for referencing that is where the file is placed. Reference column names are the columns, which exist in the generated reference text file. Generated text file is the file corresponding to which collections are to be created in MongoDB. Files path and column name (attributes of the documents) are shown into the text area.

Step 6: Once the files are generated, the next step is to load the data into MongoDB. Pentaho is used for this purpose. To integrate with pentaho, 'integrate' button is clicked. As seen in Figure 5.10, Pentaho gets opened. With the help of Pentaho, data is written into MongoDB. Files which are generated above are given as input to the pentaho. These files are in the format required by Pentaho. It will accept these files and create the collections and load the data into the collection.

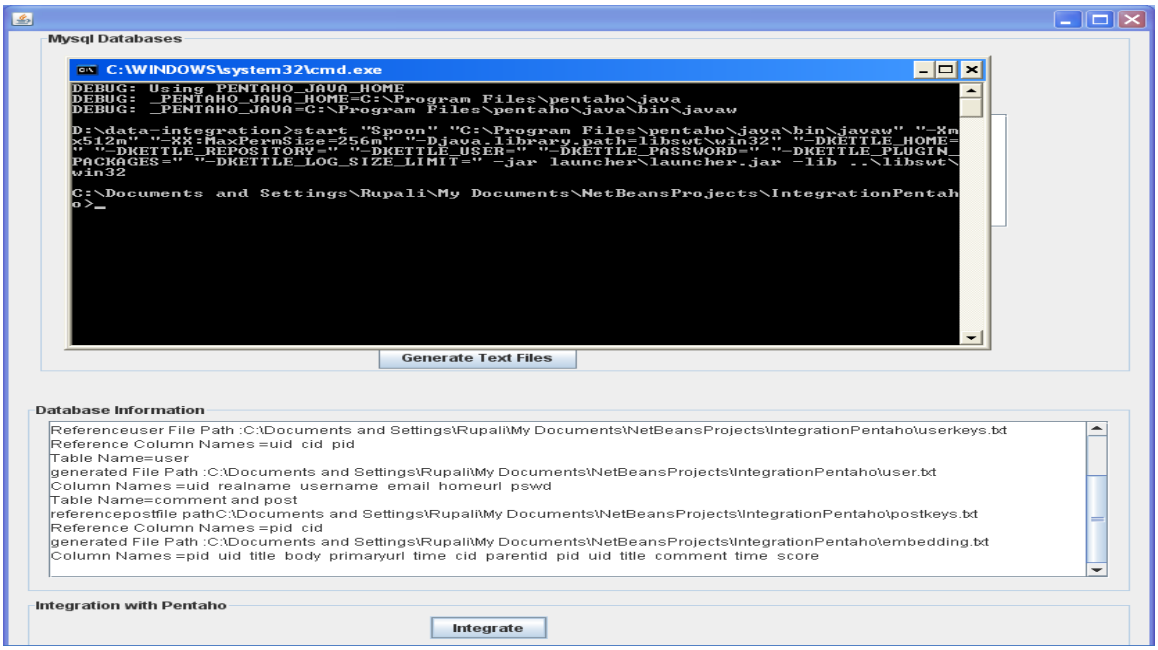

Figure 5.10: Integration with Pentaho

5.7 Post algorithm steps

Pentaho business analytics tool has been used for integration with MongoDB. It is used to write data into MongoDB. The Kettle design tool called 'Spoon' is used for integration. The steps involved in writing data into MongoDB are as follows:

Step 1: To create the collections and to load data into the collection in a database, the database should exist in MongoDB. In first step database is created in MongoDB. If

database is not created in MongoDB, Pentaho will display an error during connection establishment as shown in Figure 5.11.

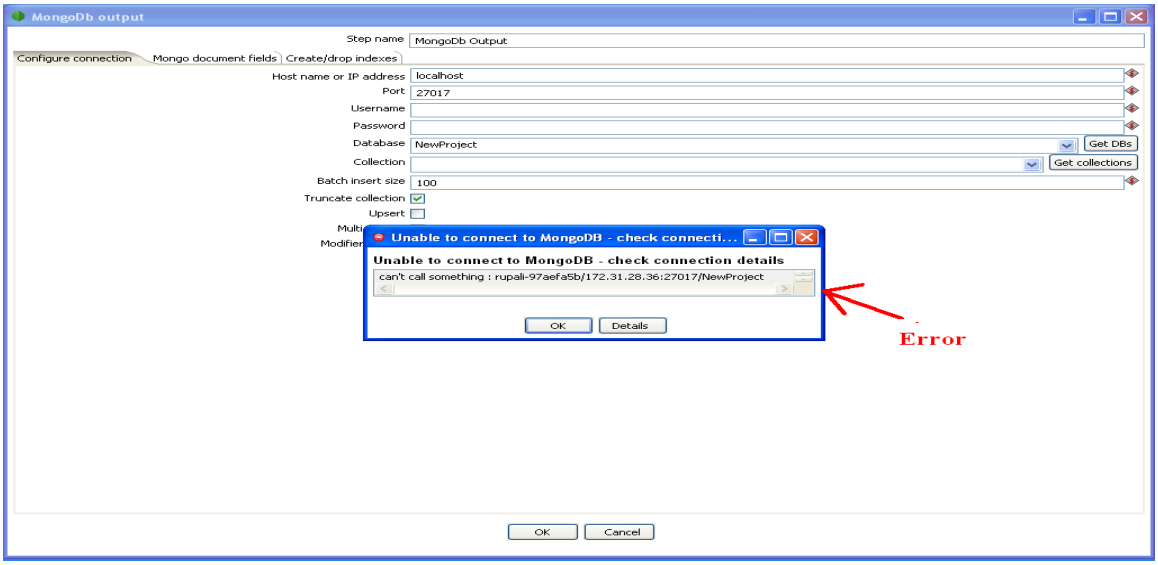

Figure 5.11: Error Window for non-existing Database in MongoDB

Step 2: Once the database is created, the next step is to load the data into database. Pentaho data integration (spoon) tool is used to execute the above task. A new transformation is created. Transformation can be created either by choosing File → New→Transformation from the menu system as shown in Figure 5.12 or by clicking on the 'New file' icon on the toolbar and choosing the 'Transformation' option.

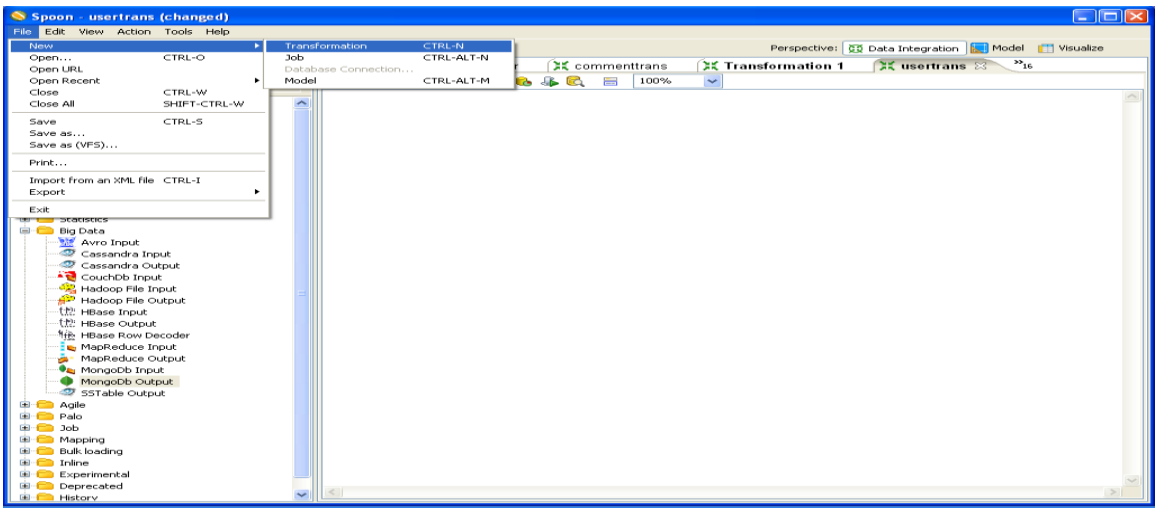

Figure 5.12: Creation of new Transformation

Step 3: After creating new transformation, the next step is to read the data from text files generated by NetBeans IDE. For this purpose, drag 'Text File Input' icon onto the transformation canvas from the input section of the design palette as shown in Figure 5.13.

.

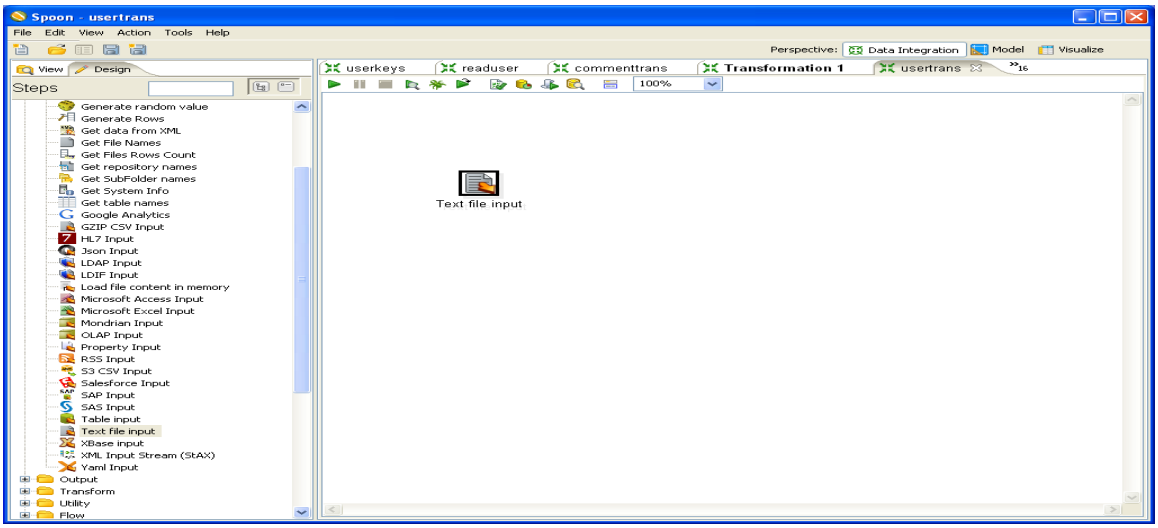

Figure 5.13: Dragging Text File Input icon on Canvas

Step 4: In this step, text file from which the data is to be read is given as input to the Pentaho. This file is the one whose corresponding collection has to be created in MongoDB. In properties of the 'Text file input' browse button is used to add the file. As shown in Figure 5.14 'user' file generated by NetBeans java API is added.

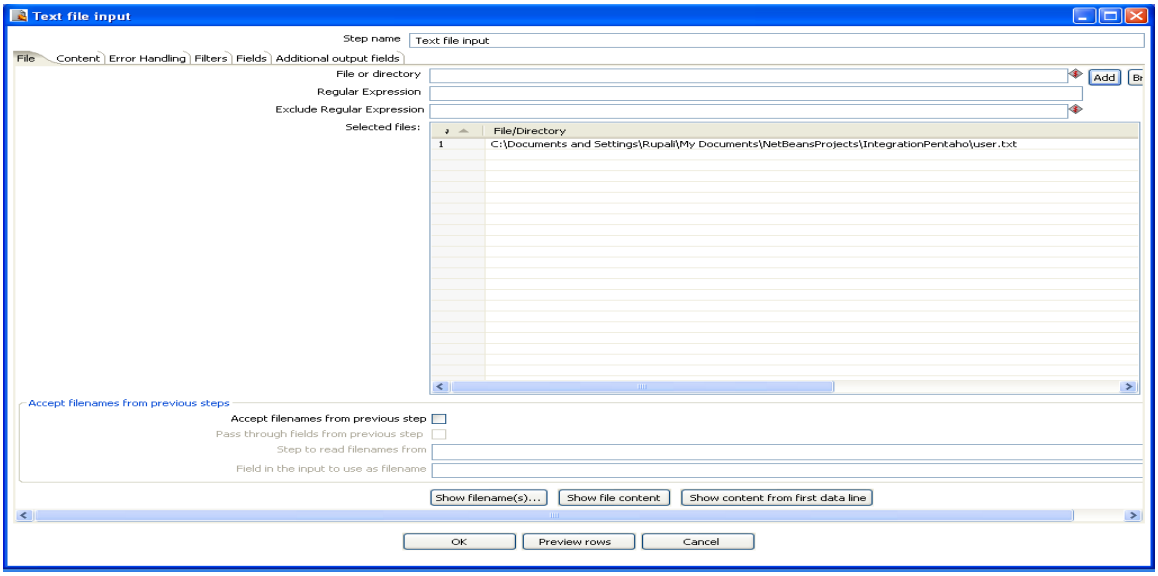

Figure 5.14: Addition of Text File

Step 5: In this step, fields from the text file are extracted. To extract the fields, click on 'Get Fields' button from fields tab. As seen in Figure 5.15, uid, realname, username, email, homeurl and pswd are fields of the documents.

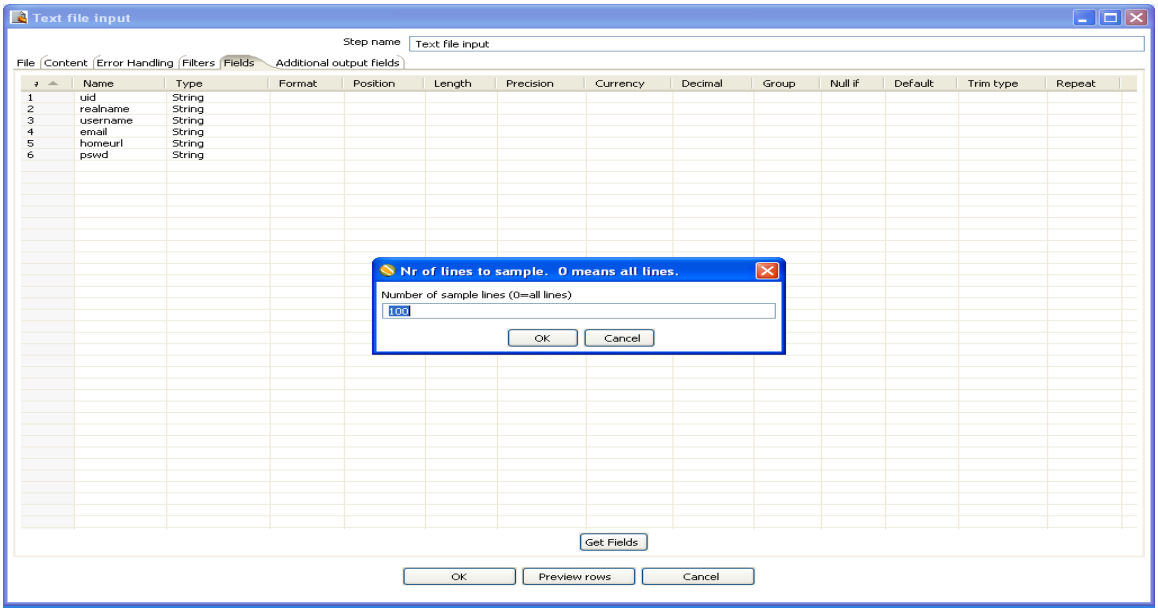

Figure 5.15: Fields Extraction from Text File

Step 6: The next step is to transfer the extracted data to the MongoDB. For this purpose, the 'MongoDB output' icon is added onto the transformation canvas from the 'Big Data' section of the design palette. Then 'Text file input' is connected with 'MongoDB Output' using output connector from 'Text file input' to 'MongoDB output' as shown in Figure 5.16.

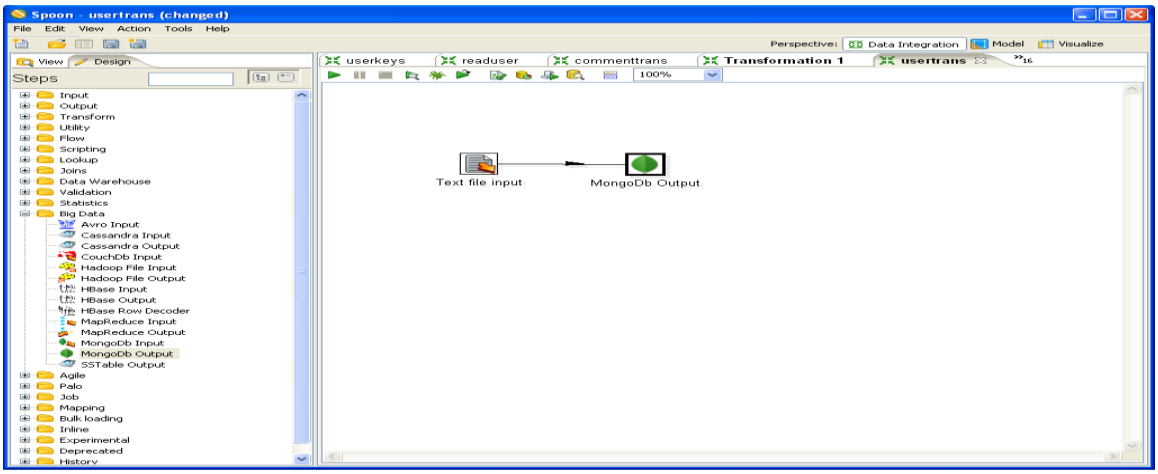

Figure 5.16: MongoDB Connectivity

Step 7: To configure the connection with MongoDB details are added in the 'Configure connection tab' in the properties of the MongoDB output. Database is the name of the already created database in MongoDB in which data is to be loaded. Collection is name of collection which is created in MongoDB. As seen in Figure 5.17

name of database is specified as test and collection name as user. The truncation option is checked to empty the collection before adding new data.

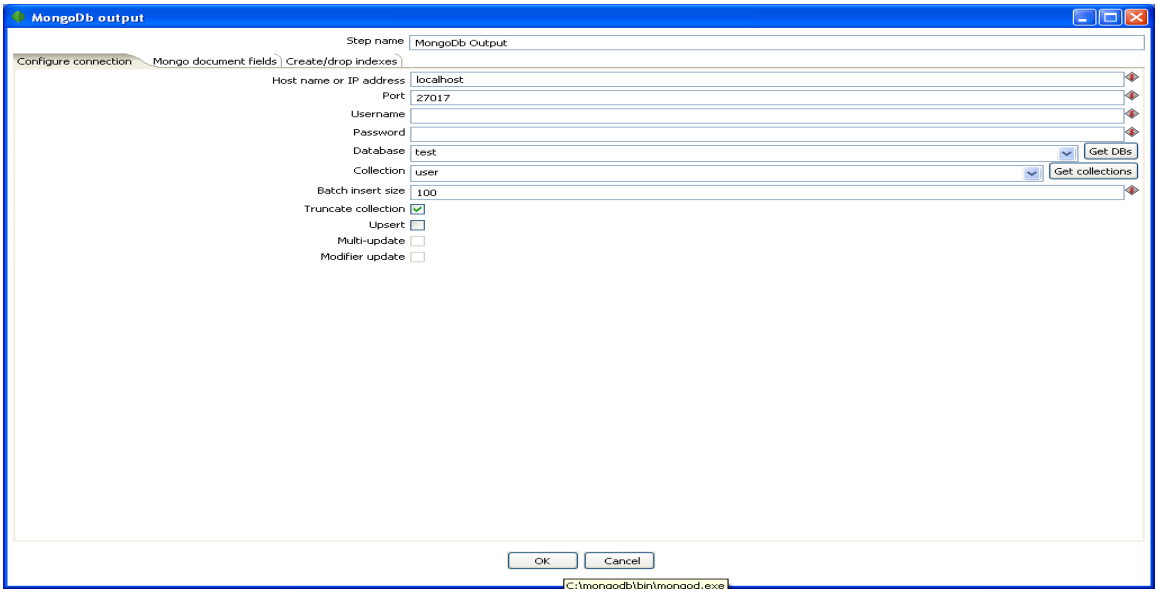

Figure 5.17: Connection Configuration

Step 8: Once the connection with MongoDB is established, the next step is to transfer the fields into the MongoDB. Fields are added into MongoDB by using 'Get fields' from the MongoDB fields' tab as shown in Figure 5.18.

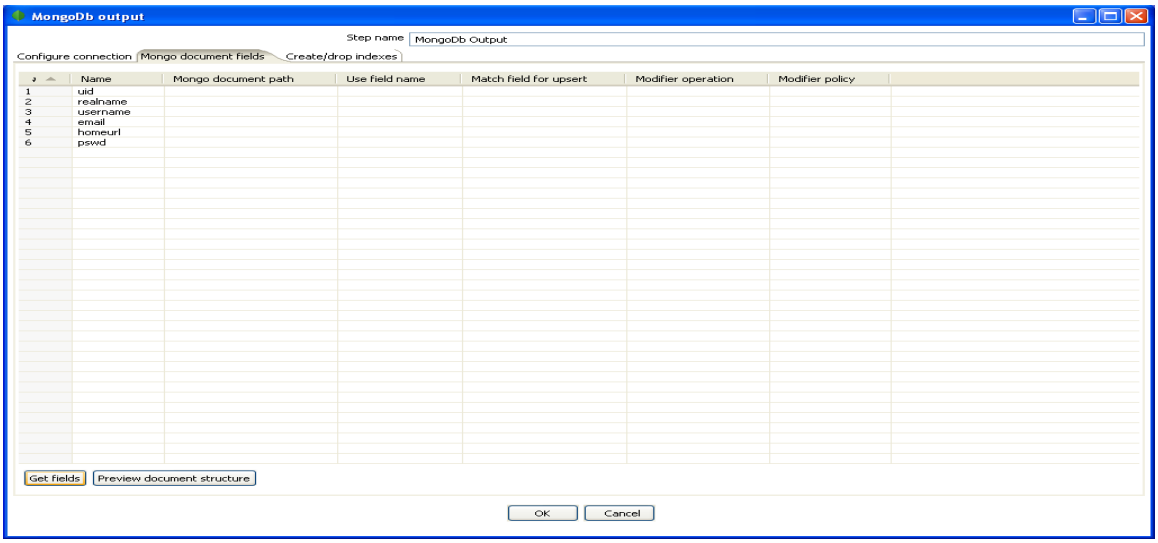

Figure 5.18: Adding fields into MongoDB

Step 9: Index can be created from the 'Create/drop indexes' tab by specifying the field name of the index to be created. In Figure 5.19 index of uid field is created.

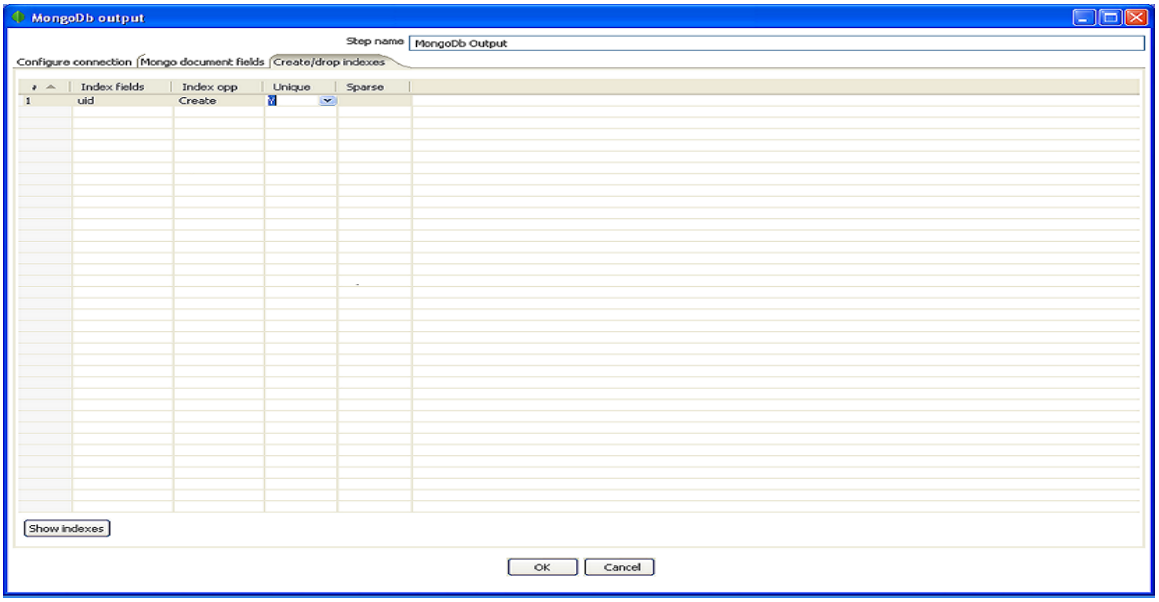

Figure 5.19: Index Creation

Step 10: Once the data is transferred, the collection is created in MongoDB by running the transformation as shown in Figure 5.20.

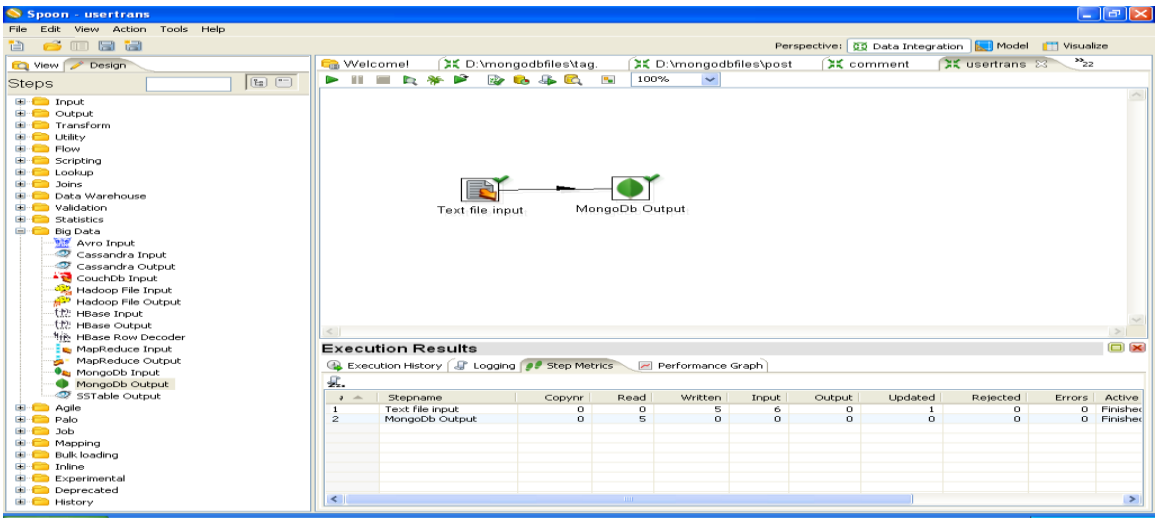

Figure 5.20: Run the Transformation

Step 11: The collection created using above steps can be checked by querying MongoDB. The queries which are used to check the collection data are as follows.

use test;

show collections;

db.user.find();

The data in the collection is shown in Figure 5.21.

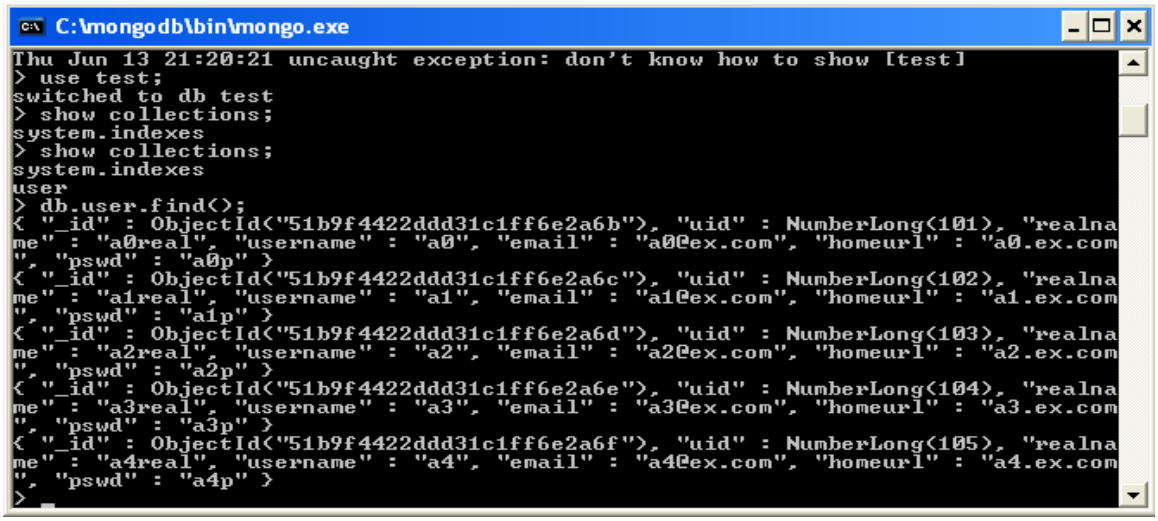

Figure 5.21: Collection created in MongoDB

5.7.1 Loading data in Embedded Document

To load the data into embedded document, first six steps are same as that for single document. Only some of the steps are different. These steps are explained in below.

Step 1: During connection establishment truncate collection, upsert, multi-update and modifier-update are checked as shown in Figure 5.22. Upsert with modifier-update enables the mode of updating. Multi-update option enables the updates to apply to all the matching documents rather than just the first. Post is the collection name.

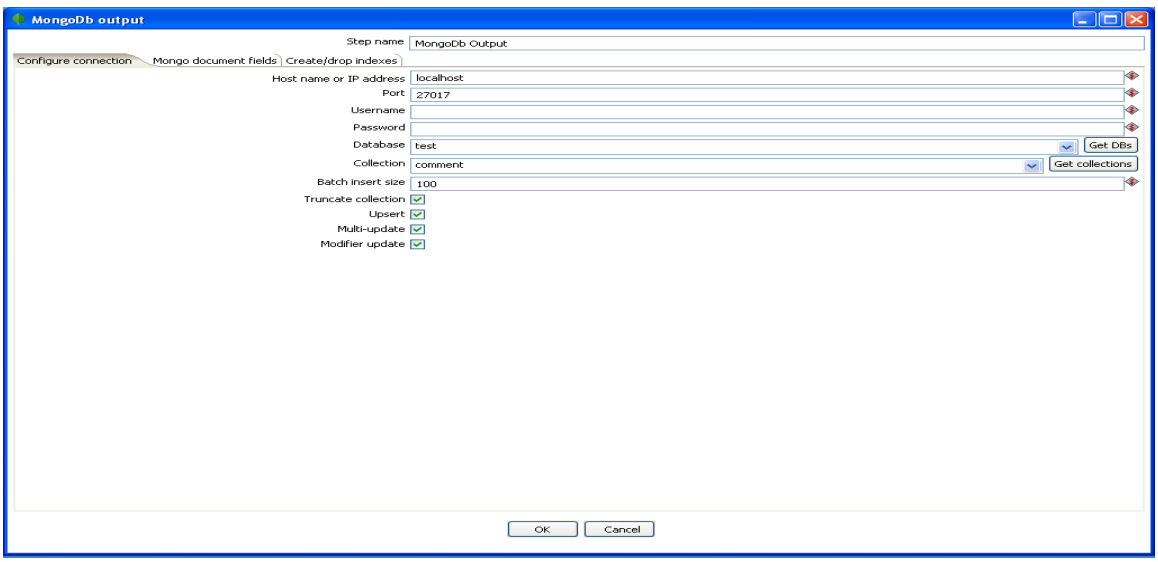

Figure 5.22: Connection Configuration for Embedded Document

Step 2: Once the connection with MongoDB is established, the next step is to transfer the fields into the MongoDB. Fields are added into MongoDB by clicking on 'Get

fields' from the 'MongoDB fields' tab as shown in figure 5.23. The 'Get fields' button populates the 'Name' column with the names of the incoming fields. In 'Mongo document path' column the hierarchical path of each field of the document is defined. For instance as shown in Figure 5.23, comments[0] is defined as the document path. The fields corresponding to comments[0] are embedded into the post collection with the array name as comments. In 'Match field for upsert' column, the fields which have to be used for matching during upsert operation are specified. When the match of the fields fails, the fields are created and inserts operation are performed. Push 'Modifier operation' specifies that corresponding fields are to be pushed into the array of the embedded documents. With 'insert & update' in the 'Modifier policy' the operations are executed whether a match exists in the collection or not. It means if match of the comments[] exists, fields are pushed into the comment array, otherwise fields are added.

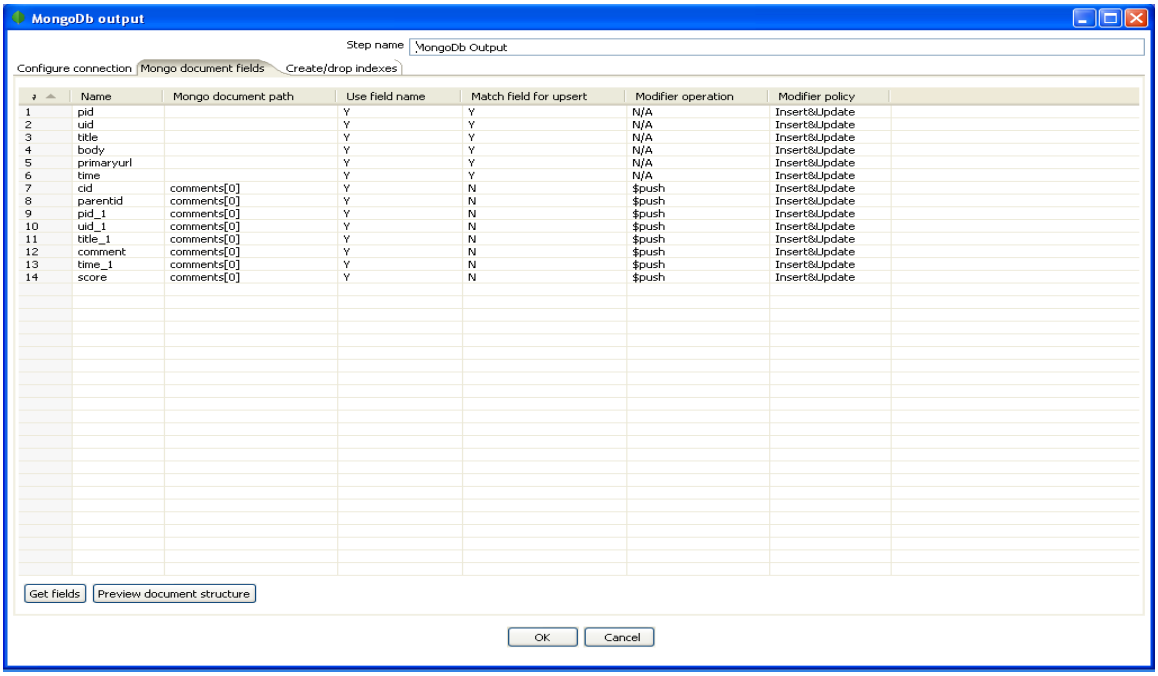

Figure 5.23: Adding Fields into MongoDB for Embedded Document

Step 3: The post collection created using above steps can be checked by querying MongoDB. As shown in Figure 5.24, the comments documents are embedded into the post collection.

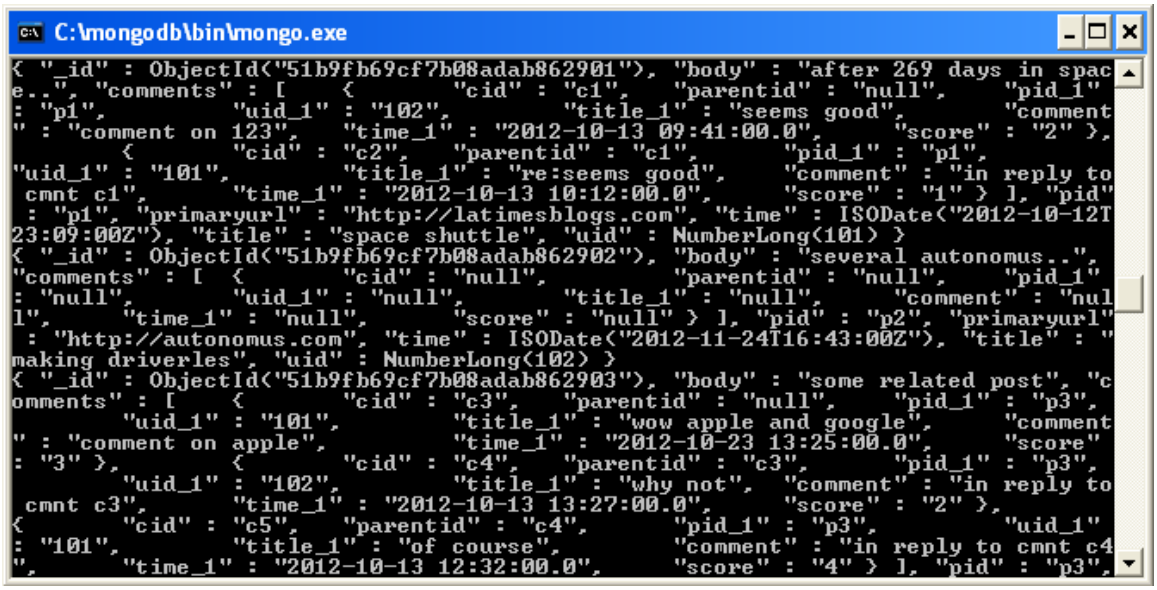

Figure 5.24: Embedded Collection created in MongoDB

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

In this thesis a data transformation approach has been proposed that takes relational database and transforms the data into MongoDB document database. The proposed technique is implemented by using NetBeans Java API with the help of Pentaho data integration tool. Transformation is divided into two steps: First step generates the text files corresponding to tables of source database, in second step these files are used and data is loaded into MongoDB.

Conclusions drawn from the work done are as follows.

- NoSQL databases have many advantages over relational databases and more and more organizations are opting for NoSQL databases.
- Among all types of NoSQL databases, document databases are more appropriate for semi-structured data storage and document databases are highly scalable.
- In comparison to another document databases MongoDB database is faster and queries in MongoDB are also simpler. So, MongoDB has broader scope than other databases.
- The proposed technique is effectively transforming data from relational database to document based database.

6.2 Future Scope

Following extensions can be done in future.

- The proposed technique transforms the data from relational database to document based NoSQL class. This work can be extended for other classes of NoSQL databases also.
- The presented work has semi-automatic post algorithm steps. The work can be extended in this aspect to make transformation fully automatic.

References

[1] Codd E. F., "A relational model of data for large shared data banks," *communications of the ACM 13*, vol. 13, no. 6, pp. 377-387 , June 1970.

[2] Finn M. A., "Fighting impedance mismatch at the database level," InterSystems Corporation, White Paper.

[3] Mazzocchi S. (2004, Feb) Stefano's Linotype. [Online]. http://www.betaversion.org/~stefano/linotype/news/46/

[4] Bondi A. B., "Characteristics of scalability and their impact on performance," in *Proceedings of the 2nd international workshop on Software and performance*, New York, NY, USA, 2000, pp. 195–203.

[5] Atwood T., "An Object-Oriented DBMS for Design Support Applications," in *Proceedings of the IEEE COMPINT 85*, 1985, pp. 299-307.

[6] Zamboulis L., Poulovassilis A. and Papamarkos G., "XML Databases," Computer Science Information System, Birkbeck College, Uni. London,.

[7] Strozzi C. Nosql a relational database management system. [Online]. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[8] Edlich S. (2009) NoSQL archive page. [Online]. http://nosql-database.org/

[9] Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshal P., Vogels W. and DeCandia G., "Dynamo: amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* , Oct., 2007, pp. 205–220.

[10] Chang F., Dean J., Ghemawat S., Hsieh W. C., Wallach D. A., Burrows M., Chandra T., Fikes A. and Gruber R. E. , "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th symposium on Operating systems design and implementation*, Berkeley, CA, USA, 2006, pp. 205–218.

[11] Borthakur D., Gray J., Sarma J. S., Muthukkaruppan K., Spiegelberg N., Kuang H., Ranganathan K., Molkov D., Menon M., Rash S., Schmidt R., Aiyer A., "Apache Hadoop goes realtime at facebook," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, New York, NY, USA, 2011, pp. 1071–1080.

[12] Vardanyan M. (2011, May) Picking the Right NoSQL Database Tool. [Online]. http://blog.monitis.com/index.php/2011/05/22/picking-the-right-nosql-database-tool/?attest=true&opt=vers

[13] Burkhart H., Rizzotti S. and Ruflin N., "Social-Data Storage Systems," in *Databases and Social Networks, Athens*, Greece, June, 2011.

[14] Higginbotham S. (2011, Sep) Sensor networks top social networks for big data. [Online]. http://gigaom.com/2010/09/13/sensor-networks-top-social-networks-for-big-data-2/

[15] Neubauer P. (2010, May) Graph Databases, NoSQL and Neo4j. [Online]. http://www.infoq.com/articles/graph-nosql-neo4j

[16] Tran T. (2012, Feb) NoSQL and Document-Oriented Databases. [Online]. http://misclassblog.com/database-design-and-development/nosql-and-document-oriented-databases/

[17] Cattell R., "Scalable Sql and NoSQL data stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12-27, May 2011.

[18] Pritchett D., "BASE: An Acid Alternative," *Queue - Object-Relational Mapping*, vol. 6, no. 3, pp. 48-55 , May 2008.

[19] Brewer E. A., "Towards robust distributed systems (abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, USA, 2000, pp. 16-19.

[20] Gilbert S. and Lynch N., "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51-59, June 2002.

[21] Arseneau D. (2010, Aug) 10 things you should know about nosql database. [Online]. http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772

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

[23] "InfiniteGraph: The Distributed Graph Database," Objectivity, Inc., Sunnyvale, White paper 2012.

[24] The memcached website. [Online]. http://www.memcached.org/

[25] Bob I. (2009, March) Drop acid and think about data. [Online]. http://bob.ippoli.to/archives/2009/04/01/pycon-2009-drop-acid-and-think-about-data/

[26] North k. (2009, August) Databases in the Cloud: Elysian Fields or Briar Patch? [Online]. http://www.drdobbs.com/database/databases-in-the-cloud-elysian-fields-or/218900502

[27] Riak. [Online]. http://basho.com/riak/

[28] Project voldemort: A distributed database. [Online]. http://www.project-voldemort.com/voldemort/

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

[30] Tokyo Cabinet: a modern implementation of DBM.. [Online]. http://fallabs.com/tokyocabinet/

[31] Protocol Buffers. Google's Data Interchange Format. [Online]. http://code.google.com/p/protobuf.

[32] Han J., Haihong E., Guan L. and Jian D., "Survey on NoSQL database," in *Pervasive Computing and applications (ICPCA) ,2011 6th Int. Conf.*, 2011, pp. 363-366.

[33] Hoff T. (2009, August) An Unorthodox Approach to Database Design : The Coming of the Shard. [Online]. http://highscalability.com/unorthodox-approach-database-design-coming-shard

[34] HyperTable. [Online]. http://hypertable.org/

[35] Lakshman A. and Malik P. , "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems*, vol. 44, no. 2, pp. 35-40 , April 2010.

[36] George G., *HBase: The Definitive Guide*.: O'Reilly Media, 2011, pp. 45-89.

[37] White T., *Hadoop: The Definitive Guide*, 2nd ed.: O'Reilly Media, 2010.

[38] Wilson J. (2008, May) Understanding HBase and BigTable. [Online]. http://architects.dzone.com/news/understanding-hbase-and-bigtab

[39] Apache CouchDB. [Online]. http://couchdb.apache.org/

[40] (2011) 10gen Inc. Agile and Scalable. [Online]. http://www.mongodb.org/

[41] Terrastore. [Online]. http://code.google.com/p/terrastore/

[42] Luciani T. J. Thrudb: Document Oriented Database Services. [Online]. http://thrudb.googlecode.com

[43] CERN – CouchDB and Ease of Use. [Online]. http://dl.couchone.com/dl/26f246a0fe23d6a53d532671331007c9/CouchDB-Case-Study-CERN.pdf

[44] Neo4j. [Online]. http://www.neo4j.org/

[45] OrientDB. [Online]. http://www.orientdb.org/

[46] AllegroGraph RDFstore. [Online]. http://www.franz.com/agraph/allegrograph/

[47] Iordanov B., "HyperGraphDB: a generalized graph database," in *WAIM'10 Proceedings of the 2010 international conference on Web-age information management* , 2010 , pp. 25-36.

[48] HyperGraphDB. [Online]. http://www.hypergraphdb.org/index

[49] Angles R., "A Comparison of Current Graph Database Models ," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference*, Arlington, VA , 2012, pp. 171 - 177.

[50] Perham M. (2009, september) Comparing Document-oriented Databases. [Online]. http://www.mikeperham.com/2009/09/01/comparing-document-oriented-databases/

[51] MongoDB Production Deployments. [Online]. http://www.mongodb.org/about/production-deployments/

[52] Widenius M. and Axmark D., "MySQL Introduction," *Linux Journal*, vol. 1999, no. 67, November 1999.

[53] Bleuel J. (2012, May) Kettle and NoSQL: MongoDB. [Online]. http://kettle.bleuel.com/2012/05/23/kettle-and-nosql-mongodb/

List of Publications

[1] Rupali Arora and Rinkle Rani, "Implementation of Relational Queries in Graph Database," *International Journal on Information and Communication Technologies*, vol. 6, no. 2, pp. 354-357, June 2013.

[2] Rupali Arora and Rinkle Rani, "Modeling and Querying Data in MongoDB," *International Journal of Scientific and Engineering Research (IJSER)*, vol. 4, no. 7, July 2013.

[3] Rupali Arora and Rinkle Rani, "An Algorithm for Transformation of Data from MySQL to NoSQL (MongoDB)," *International Journal of Advanced Studies in Computers, Science and Engineering (IJASCSE)* **[Communicated].**

Appendix

Pseudo code for transformation of data from MySQL to MongoDB

Global Parameters passed

i. An array for storing the table names of source database.
ii. An array to store the table names which are to be embedded.
iii. A variable to store the embedded table.
iv. An array to store column names of a table.
v. An array to store column values of all columns.
vi. An array to store primary key/composite key of a table.
vii. An array for storing exported table names (to which the foreign keys refer.
viii. An array for storing primary keys of exported tables.
ix. An array to store column names of embedded table on the basis of which Join has to be performed.

Function Name: **Main()**

Called By: Operating System

Calling:

i. get_table_names()
ii. get_embed_table_names()
iii. get_embedded_table()
iv. get_column_names()
v. get_column_values()
vi. get_primary_keys()
vii. get_exported_keys_table_names()
viii. check_for_null()
ix. get_exported_tables_primary_keys()
x. generate_tbl_text_files()
xi. generate_tbl_keys_text_files()
xii. get_col_names_for_join()

xiii. integrate_with_pentaho()

Parameters Passed: Source database

Return: Target database

Function Name: **get_table_names()**

Called By: Main()

Calling: Nothing

Parameters Passed: Source database

Purpose: To extract all the table names from source database.

Function Body:

i. Find all the table names.

ii. Stores table names in an array.

iii. Return to main() function.

Return: An array of table names.

Function Name: **get_embed_table_names()**

Called By: Main()

Calling: Nothing

Parameters Passed: Array of table names

Purpose: To ask user the table names those are to be embedded in MongoDB.

Function Body:

i. Ask users about tables to be embedded.

ii. Stores table names in an array.

iii. Return to main() function.

Return: An array of embedded table names.

Function Name: **get_embedded_table()**

Called By: Main()

Calling Nothing

Parameters Passed: Array of embedded table names

Purpose: To ask user the table name in which other table is to embed.

Function Body:

i. Ask user about embedded table.
ii. Store table name in a variable.
iii. Return to main() function.

Return: embedded table name

Function Name: **get_column_names()**
Called By: Main()
Calling: Nothing
Parameters Passed: Table name
Purpose: To extract the column names of the table
Function Body:

i. Find all the column names of the table.
ii. Store the column names in an array.
iii. Return to main() function.

Return: An array of column names.

Function Name: **get_column_values()**
Called By: Main()
Calling: Nothing
Parameters Passed: Column Name
Purpose: To extract the column values of the columns.
Function Body:

i. Find the values of the all the columns.
ii. Store the values in array
iii. Return to main() function.

Return An array of column values.

Function Name: **get_primary_keys()**
Called By: Main()
Calling: Nothing
Parameters Passed: Table Name
Purpose: To get primary key/Composite primary key
Function Body:

i. Find primary key/composite key of the table by querying MySQL database.
ii. Store the values in an array.
iii. Return to main() function.

Return: An array of the primary keys/composite keys.

Function Name: **get_exported_keys_table_names()**
Called By: Main()
Calling: Nothing
Parameters Passed: Table Name
Purpose: To get table names to which exported keys of the table belongs (For referencing in MongoDB).
Function Body:

i. Find the table name to which the foreign key refers by firing SQL queries.
ii. Store all the table names in an array.
iii. Return to main() function.

Return: An array of the table names (for referencing).

Function Name: **check_for_null**
Called By: Main()
Calling: Nothing
Parameters Passed: Table Names
Purpose: To check whether tables has referential integrity constraints or not
Function Body:

i. Check whether the exported keys table array is empty or not
ii. Store the value (0 or 1) in Boolean variable.
iii. Return to the main() function

Return: A Boolean variable

Function Name: **get_exported_tables_primary_keys**
Called By: Main()
Calling: Nothing

Parameters Passed: Table Names

Purpose: To get the primary keys of the tables to which the table refers.

Function Body:

i. Find the primary key of the tables (to which the foreign keys of tables refers).

ii. Store all the primary keys in an array.

iii. Return to main() function.

Return: An array of primary keys.

Function Name: **generate_tbl_txt_files()**

Called By: Main()

Calling: Nothing

Parameters Passed: Column Names, Column Values

Purpose: To generate the text file in the format understood by Pentaho

Function Body:

i. Generate text files having name as that of table name and contains Column Name and their values in a specific format.

ii. Return to main() function.

Return: Text files

Function Name: **generate_tbl_keys_text_files()**

Called By: Main()

Calling: Nothing

Parameters Passed: Primary keys of table, primary keys of exported tables

Purpose: To generate text files which are used for referencing in MongoDB in the format understood by Pentaho

Function Body:

i. Generate text file in a specific format having name as that of table name and contains its primary key and the primary key of the table to which this table is related.

ii. Return to main() function

Return	Text files for referencing

Function Name:	**get_col_for_join()**
Called By:	main()
Calling:	Nothing
Parameters Passed:	Array of embedded table names
Purpose:	To get the column names on the basis of which Join is to be performed (Join of the tables which are to be embedded).

Function Body:

i. Ask the user about the columns of the embedded table which are to be used for the Join.
ii. Store the column names into the array.
iii. Return to main() function.

Return: An array of the column names.

Function Name:	**integrate_with_pentaho()**
Called By:	main()
Calling:	Nothing
Parameters Passed:	Text files
Purpose:	Write data into MongoDB

Function Body:

i. Take the text files as input.
ii. Connect to the MongoDB.
iii. Create collection in MongoDB
iv. Load the data into collection.

Return: Nothing