

# DATA MIGRATION FROM STANDARD SQL TO NoSQL

A Thesis Submitted to the College of  
Graduate Studies and Research  
In Partial Fulfillment of the Requirements  
For the Degree of Master of Science  
In the Department of Computer Science  
University of Saskatchewan  
Saskatoon

By

MUHAMMAD MUGHEES

© Copyright Muhammad Mughees, November, 2013. All rights reserved.

### Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science  
176 Thorvaldson Building  
110 Science Place  
University of Saskatchewan  
Saskatoon, Saskatchewan (S7N 5C9)  
Canada

## ABSTRACT

Currently two major database management systems are in use for dealing with data, the Relational Database Management System (RDBMS) also known as standard SQL databases and the NoSQL databases. The RDBMS databases deal with structured data and the NoSQL databases with unstructured or semi-structured data. The RDBMS databases have been popular for many years but the NoSQL type is gaining popularity with the introduction of the internet and social media. Data flow from SQL to NoSQL or vice versa is very much possible in the near future due to the growing popularity of the NoSQL databases.

The goal of this thesis is to analyze the data structures of the RDBMS and the NoSQL databases and to suggest a Graphical User Interface (GUI) tool that migrates the data from SQL to NoSQL databases. The relational databases have been in use and have dominated the industry for many years. In contrast, the NoSQL databases were introduced with the increased usage of the internet, social media, and cloud computing. The traditional relational databases guarantee data integrity whereas high availability and scalability are the main advantages of the NoSQL databases. This thesis presents a comparison of these two technologies. It compares the data structure and data storing techniques of the two technologies. The SQL databases store data differently as compared to the NoSQL databases due to their specific demands. The data stored in the relational databases is highly structured and normalized in most environments whereas the data in the NoSQL databases are mostly unstructured. This difference of the data structure helps in meeting the specific demands of these two systems. The NoSQL DBs are scalable with high availability due to the simpler data model but does not guarantee data consistency at all times. On the other hand the RDBMS systems are not easily scalable and available at the same time due to the complex data model but guarantees data consistency. This thesis uses CouchDB and MySQL to represent the NoSQL and standard SQL databases respectively. The aim of the

research in this document is to suggest a methodology for data migration from the RDBMS databases to the document-based NoSQL databases.

Data migration between the RDBMS and the NoSQL systems is anticipated because both systems are currently in use by many industry leaders. This thesis presents a Graphical User Interface as a starting point that enables the data migration from the RDBMS to the NoSQL databases. MySQL and CouchDB are used as the test databases for the relational and NoSQL systems respectively. This thesis presents an architecture and methodology to achieve this objective.

## ACKNOWLEDGMENTS

I would like to take this opportunity to thank my supervisor, Dr. Ralph Deters whose encouragement, continuous guidance and deep knowledge about the thesis topic kept me motivated and focused during the research. I would also like to thank Dr. Julita Vassileva who helped me with the initial idea and provided valuable information about the system I have developed for this research topic. My wife was instrumental in supporting me and it would have not been possible without her help and support.

# TABLE OF CONTENTS

page

<u>ABSTRACT.....</u>	<u>ii</u>
<u>LIST OF TABLES.....</u>	<u>ix</u>
<u>LIST OF FIGURES.....</u>	<u>x</u>
<u>LIST OF ABBREVIATIONS.....</u>	<u>xiii</u>
<u>INTRODUCTION.....</u>	<u>1</u>
1.1 Rationale for data migration.....	1
1.2 Overview.....	3
1.2.1 Key-Value Stores.....	4
1.2.2 Wide Column Store / Column Families.....	5
1.2.3 Document Stores:.....	5
1.2.4 Graph Models:.....	6
<u>PROBLEM DEFINITION.....</u>	<u>8</u>
2.1 Introduction to Data Migration from Relational SQL to NoSQL databases.....	8
2.2 Description.....	9
2.3 Research Goal.....	11
2.3.1 Goal one: Migration.....	11
2.3.2. Goal two: Identical Functionalities.....	11
2.3.3 Goal three: Performance.....	12
2.3.4 Goal four: Code Comparison.....	12
<u>LITERATURE REVIEW.....</u>	<u>13</u>
3.1 Relational Models.....	14
3.1.1 Primary Key Concept.....	17
3.1.2 Foreign Key.....	17
3.1.3 Normalization in the relational model.....	18
3.1.3.1 First Normal Form (1NF).....	18
3.1.3.2 Second Normal Form (2NF).....	19
3.1.3.3 Third Normal Form (3NF).....	19
3.1.4 Process of Normalization.....	19
3.1.5 Data integrity and reliability.....	20
3.1.5.1 Atomicity.....	21
3.1.5.2 Consistency.....	21

3.1.5.3 Isolation .....	21
3.1.5.4 Durability.....	21
3.2 Structured Query Language (SQL) .....	23
3.3 NoSQL databases .....	27
3.3.1 Key-Value Stores .....	30
3.3.2 Columnar Databases: .....	32
3.3.2.1 Column .....	32
3.3.2.2 SuperColumn.....	33
3.3.2.3 Column Family.....	33
3.3.3 Document Store (DS).....	36
3.3.4 Graph Model: .....	39
3.3.5 MapReduce .....	42
3.3.6 CAP Theorem .....	44
3.4 Data Migration .....	47
3.4.1 Choosing a NoSQL database for the Conference System .....	50
3.5 Conclusion.....	55
<b><u>DATA MIGRATION ARCHITECTURE .....</u></b>	<b><u>57</u></b>
4.1 Types and scenarios in relational database.....	58
4.1.1 Table with primary key .....	58
4.1.1.1 Single Column primary key .....	58
4.1.1.2 Composite primary key .....	59
4.1.2 Tables with a single foreign key .....	59
4.1.3 Tables with multiple foreign keys.....	60
4.1.4 One-to-one relationship .....	61
4.1.5 One-to-many relationship .....	61
4.1.6 Many-to-many relationship.....	62
4.2 Migrating from relational MySQL to NoSQL CouchDB .....	62
<b><u>TEST BED: CONFERENCE SYSTEM.....</u></b>	<b><u>66</u></b>
5.1 Introduction to the Conference System (CS) .....	66
5.1.1 System requirements and workflow.....	67
5.2 RDBMS Conference System Data Model.....	68
5.3 CouchDB Data Model for Conference System .....	70
5.3.1 Views .....	72
5.3.1.1 Permanent Views.....	73
5.3.1.2 Temporary Views .....	73
<b><u>IMPLEMENTATION.....</u></b>	<b><u>74</u></b>
6.1 Data Migration Steps .....	75
<b><u>EVALUATION.....</u></b>	<b><u>86</u></b>
7.1 Evaluation Goals .....	86
7.1.1 Has the data been migrated successfully to CouchDB?.....	86

7.1.2 Can the same operation be performed in both systems and achieve the same result? .87	87
7.1.3 What is the speed difference for performing the same operation in both systems? ....87	87
7.1.4 How complex is the code for the same operation in both systems? .....87	87
7.2 Experiment Goals.....88	88
7.2.1 Has the data migrated successfully to CouchDB? .....88	88
7.2.1.1 Data migration from single table in multiple CouchDB documents .....88	88
7.2.1.2 Data migration from multiple tables in multiple CouchDB documents.....90	90
7.2.2 Can the same operation be performed in both systems and achieve the same result? .90	90
7.2.3 What is the speed difference for same operation performed on both systems? .....91	91
7.2.4 How complex is the code for the same operation in both systems? .....91	91
<b>RESULTS .....92</b>	<b>92</b>
8.1 Has the data migrated successfully?.....92	92
8.1.1 Data population and extraction in MySQL.....92	92
8.1.2 Data migration into the CouchDB .....93	93
8.1.2.1 Data migration from single MySQL table into CouchDB documents .....93	93
8.1.2.2 Data migration from multiple MySQL tables into the CouchDB documents.....96	96
8.2 Can the same operation be performed in both systems and achieve the same results?.....98	98
8.2.1 Verify data after SELECT operation .....98	98
8.2.2 Verify data after the UPDATE operation .....100	100
8.2.3 Verify data after DELETE operation.....102	102
8.3 Compare the speed of various operations.....103	103
8.3.1 SELECT operation time comparison.....104	104
8.3.2 INSERT operation time comparison.....106	106
8.3.3 UPDATE operation time comparison.....108	108
8.3.4 DELETE operation time comparison.....111	111
8.4 Determine the code complexity.....114	114
<b>CONCLUSION AND FUTURE WORK .....119</b>	<b>119</b>
9.1 Conclusion.....119	119
9.2 Future Work .....124	124
<b>LIST OF REFERENCES .....128</b>	<b>128</b>
<b>DATA MODEL OF CONFERENCE SYSTEM .....133</b>	<b>133</b>
A1. Users.....133	133
A2. Conferences.....133	133
A3. Documents .....133	133
A4. Doc_author.....133	133
A5. Reviewed_documents .....133	133
A6. Review_status .....133	133
A7. Reviewed_doc_hierarchy.....134	134
A8. Assign_doc.....134	134
A9. Assignments .....134	134



A10. Comments .....	134
A11. Roles.....	134
A12. User_role.....	134
A13. Survey .....	134
A14. Evaluation_by_author .....	135
A15. Assignment_doc.....	135
A16. Grades .....	135

## LIST OF TABLES

<u>Table</u>	<u>page</u>
Table 3 - 1: Relational Model terminologies and explanation.....	16
Table 3 - 2: NoSQL distinguishing properties.....	30
Table 3 - 3: Characteristics of various NoSQL databases .....	52
Table 6 - 1: A record in MySQL.....	81
Table 8 - 1: SELECT operation times comparison in tabular form .....	104
Table 8 - 2: INSERT operation times comparison in tabular form .....	107
Table 8 - 3: UPDATE operation times comparison in tabular form.....	109
Table 8 - 4: DELETE operation times comparison in tabular form .....	111
Table 8 - 5: Comparative study of code complexity.....	115

## LIST OF FIGURES

<u>Figure</u>	<u>page</u>
Figure 2 - 1: Data storage in RDBMS.....	9
Figure 2 - 2: Data storage in NoSQL.....	10
Figure 2 - 3: How data migrate from RDBMS to NoSQL.....	11
Figure 3 - 1: Illustrates Data Model Structure .....	15
Figure 3 - 2: An illustration of Foreign Key.....	17
Figure 3 - 3: Possible process flow of normalization .....	20
Figure 3 - 4: Key-Value document structure in CouchDB .....	31
Figure 3 - 5: Columnar database storage structure .....	36
Figure 3 - 6: Same data representation stored in RDBMS and columnar structure .....	36
Figure 3 - 7: Documents stored document store.....	38
Figure 3 - 8: Document structure in JSON format.....	38
Figure 3 - 9: Graph and basic components .....	40
Figure 3 - 10: Graph traversing .....	41
Figure 3 - 11: Usage of index in Graph database .....	42
Figure 3 - 12: An overview of MapReduce process .....	44
Figure 3 - 13: An illustration of CAP theorem and NoSQL DBs compliance.....	47
Figure 3 - 14: MVCC locking mechanism in CouchDB .....	54
Figure 4 - 1: A table with single column primary key.....	59
Figure 4 - 2: Relational table with composite primary key .....	59
Figure 4 - 3: Table with single foreign key .....	60

Figure 4 - 4: Table with multiple foreign keys.....	61
Figure 4 - 5: Data Migration Architecture.....	64
Figure 5a - 1: Conference System Data Model (part 1) .....	69
Figure 5b - 1: Conference System Data Model (part 2) .....	70
Figure 5 - 1: Simplest form of key-value pair.....	71
Figure 5 - 2: Object as complex component of key-value structure.....	72
Figure 5 - 3: An array as complex component of key-value structure.....	72
Figure 6 - 1: High level implementation architecture.....	75
Figure 6 - 2: PHP code with embedded structure of the CouchDB document.....	76
Figure 6 - 3: Mapping RDBMS table with CouchDB document.....	79
Figure 6 - 4: Structure of a CouchDB document.....	80
Figure 6 - 5: MySQL record as CouchDB document after data migration .....	82
Figure 6 - 6: Structure of USERS table in MySQL .....	83
Figure 6 - 7: Structure of DOCUMENTS table in MySQL .....	83
Figure 6 - 8: Structure of REVIEWED_DOCUMENTS table in MySQL.....	83
Figure 6 - 9: Structure of REVIEW_STATUS table in MySQL.....	84
Figure 6 - 10: Mapping multiple table fields in CouchDB document.....	85
Figure 7 - 1: Data Migration Process flow.....	89
Figure 8 - 1: Data migration tool.....	94
Figure 8 - 2: CS form showing data from single MySQL table.....	94
Figure 8 - 3: Migrated data from USERS table inside CouchDB documents .....	95
Figure 8 - 4: Documents assigned to reviewers in MySQL.....	96
Figure 8 - 5: Documents assigned to reviewers in CouchDB .....	97
Figure 8 - 6: Documents and author information in MySQL.....	99
Figure 8 - 7: Documents and authors information in CouchDB.....	100

Figure 8 - 8: CouchDB Users' information.....	101
Figure 8 - 9: CouchDB record after update operation .....	102
Figure 8 - 10: Users' information in CouchDB after the DELETE operation .....	103
Figure 8 - 11: SELECT operation time comparison .....	106
Figure 8 - 12: INSERT operation time comparison .....	108
Figure 8 - 13: UPDATE operation time comparison .....	110
Figure 8 - 14: DELETE operation time comparison .....	113
Figure 8 - 15: Code complexity comparison.....	116

## LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
BASE	Basically Available, Soft state, Eventually consistent
CAP	Consistency, Availability, and Partition tolerance
CC	Cloud Computing
CDB	CouchDB
DB	Database
DBMS	Database Management System
DM	Data Migration
DMT	Data Migration Tool
DC	Distributed Computing
DS	Data Store
GUI	Graphical User Interface
IT	Information Technology
JSON	Java Script Object Notation
NoSQL	Not Only SQL
RDBMS	Relational Database Management System
REST	Representational State Transfer
SQL	Structured Query Language
XML	Extensible Markup Language

## CHAPTER 1

### INTRODUCTION

#### **1.1 Rationale for data migration**

The introduction of the internet, social web sites, and cloud computing has challenged the supremacy of the relational databases as the ultimate choice of database management system. Factors such as cost, volume of data, variety of data and the pace at which the data is being created and consumed play an important role in deciding how and where the data should be stored and managed.

Features like high availability, scalability, and replication are all available in the traditional relational databases but they usually come with a high cost. The main factors for the popularity of the NoSQL databases include cheap hardware, easy scalability, and high availability.

Most of the data that comes from social web sites, clouds, and mobile phones is unstructured with varying nature. The data created and consumed using these resources is also huge in volume and requires regular scalability with high availability. These tasks can still be achieved with the traditional RDBMS databases but at a high cost which makes the NoSQL databases the database of choice as they are cheaper comparatively.

The general impression is that only big social web sites like Facebook, Twitter, Google, and Foursquare are concerned due to their specific requirements but that is not totally true. The Web sites already using the RDBMS for storing and managing the data will start to feel the heat when the number of users and data will starts growing beyond expectations and thus makes it hard to keep up with the data requirement. The key factors for the popularity of NoSQL as compared to the RDBMS systems are as follows:

- Low cost
- High performance

- Easily Scalable
- Highly available

The NoSQL approach takes advantage of all the above mentioned factors at the cost of data consistency. Due to compliance to the Atomicity, Consistency, Isolation, Durability (ACID) principles, the RDBMS systems guarantee data integrity and consistency at all times but they are not easily scalable. NoSQL systems compromise data consistency to achieve better performance, scalability and high availability. For applications where data consistency is not important the RDBMS systems does not necessarily remain a good choice. “Relational database technology has served us well for 40 years, and will likely continue to do so for the foreseeable future to support transactions requiring ACID guarantees. But a large, and increasingly dominant, class of software systems and data do not need those guarantees,” [49]. Thus organizations running databases currently on the RDBMS systems may feel a need to migrate from RDBMS to NoSQL databases when data grows beyond the capacity of one server and costly vertical scaling is required. The same situation is expected with a distributed RDBMS system with growing data issues.

Considering the factors discussed above it is safe to assume that there will be future data migration requirements from the relational to the NoSQL databases. Foreseeing this future requirement, this thesis focuses on data migration approach from the RDBMS to the NoSQL databases.



## 1.2 Overview

Over the years the relational database management systems (RDBMS) have been the leading technology in the IT world. The RDBMS has dominated the industry since its inception when E. F. Codd introduced it in 1970 [15]. Several different approaches have been in practice besides the relational concept. The hierarchical and the network data models were introduced in 1960 and implemented by “Integrated Data Store of Honeywell (network model) and Information Management System (IMS) by IBM (Hierarchical Model)” [48]. Being dominant the RDBMS systems embraced and merged other prevailing ideas. The object-based data model was merged with the RDBMS to form the object-relational model.

This approach of using the relational systems in all scenarios came under scrutiny with the introduction of the web. The relational data model works well with the traditional applications where the data is not massive and distributed. Despite being on top for several years the capability of the RDBMS for processing and handling large amount of data remains in question. However, handling and processing huge amounts of data is almost mandatory for big organizations such as Google, Amazon, Twitter, LinkedIn and Facebook. In short, the RDBMS systems are not the suitable candidate for processing the large amounts of data where the data is distributed over a network in a cluster of servers or over a grid in different geographical locations. A new approach under the flag of the NoSQL database was introduced to mitigate some of the problems not handled elegantly by the RDBMS. In 1998 Carl Strozzi first time used the term NoSQL for his open source relational database "Strozzi NoSql" that did not use the Structured Query Language (SQL) to access and manipulate the data [51]. Instead they used Application Programming Interface (API) with various plugins and libraries or RESTful API with Hypertext Transfer Protocol (HTTP) protocol. This is the reason why they are called “NoSQL” databases.

NoSQL is a database management system that is entirely different from the RDBMS in many ways. The primary reason for using the NoSQL DBs is their flexibility, scalability, speed, and high availability. For these reasons NoSQL is making its space in the industry, however, it is still not mature enough and lacks 'standards'. "For any of the new databases, you may have Pig, Hive, SPARQL, Mongo Query Language, Cypher, or others. These languages have little in common." [37]. The NoSQL system meets the requirements of the corporations who deal with terabytes of data. The data is mostly stored in a distributed environment and does not need schemas or join operations to retrieve data. There is no concept of primary or foreign keys in NoSQL since the data is not stored in tables and is not relational. The data in the NoSQL CouchDB is stored in the form of documents. Each document is similar to a row in a table and the group of documents represents a logical table. It takes the advantage of horizontal scaling to improve performance by adding new machines in the system and distributing the additional load equally.

Since the introduction of the NoSQL idea, numerous databases have been designed depending on the requirements of these systems. These databases have been categorized into different groups. The next section discusses and compares the current NoSQL designs in use by the Information Technology (IT) industry.

The category of the NoSQL DBs depends on how the data is stored. Currently the NoSQL DB is categorized in four major categories [52] as follows:

### **1.2.1 Key-Value Stores**

Key-Value Stores allow applications to store data in a schema-free environment such as a hash table. Hash tables contain a value for a particular key. The data consists of a key as a string and real data which forms the value part in the pair. The data type of stored data depends on the

programming language and it can be any data type supported by the language. Examples of data types are string, integer, array or object. The data stored in such fashion alleviates the need of a fixed data model and structured data. Amazon was the first to implement the key value structure to resolve their data availability and scalability issues using the Dynamo storage system [23]. Key-Value Stores are useful in environments with distributed hash tables and caching which Dynamo has implemented to improve high read performance and "consistent hashing to partition its key space across its replicas and to ensure uniform load distribution" [23].

### **1.2.2 Wide Column Store / Column Families**

A Wide Column Store also uses a key-value pair but the key is two dimensional. In this type of structure a column key and a row key is required to access a stored value. A timestamp can also be added as part of the key as practiced in Google's Big Table [13]. This type of model is suitable for huge amounts of data in distributed environment as implemented by Google's Big table. Facebook's messenger services also use Hbase to support billions of messages per day [5].

### **1.2.3 Document Stores:**

As the name suggests Document Stores consists of documents. This model is very useful for horizontal scalability which allows the addition of cheap commodity servers or resources from the cloud as the database grows. Document Stores are similar to the Key-Value Stores and takes advantage of hashed tables. The key-value pair is stored in distributed hash tables (DHT) in hash buckets. An index is created on the key-value pairs in these buckets for search purposes. The hash value is a unique number generated from the data. The algorithm for creating hash values distributes the key-value pairs evenly among hash buckets [50]. This means that if the DHT have 4 hash buckets and 20 key-value pairs then each hash bucket will have 5 key-value pairs.

This model makes the parallelization very easy with the addition of more hash buckets when more servers are added due to data growth.

The Document Stores are useful for web-based applications with data distributed in a network or over a grid.

#### **1.2.4 Graph Models:**

Graph DBs consist of nodes, relationships among nodes and their properties. Graph models are more easily scalable over many servers than SQL DBs. The main feature of a graph database is its improved ability to handle relationships compared to a relational model. The Graph DB traverses through huge numbers of edges in a fraction of the time of a relational join due to direct links between nodes [30]. The graph DBs uses the concept of vertex (node), edge (relationship) and property. Graph DBs stores data with nodes and edges using a graph algorithm. This model is useful where the data model is recursive. An example of these attributes is as follows:

Node: Employee

Property: first name, last name, designation and department

Relationship: Employee part of a department

The Graph databases are useful for social networking web sites such as Twitter that uses FlockDB to store the graph data about relationships between users, information about who's following whom and more [35].

Due to the increased popularity of the NoSQL DBs and the existing wide usage of the RDBMS, the possibility of future data migration between them is highly likely. The task of data migration is difficult due to the entirely different data model. Information in the relational databases is structured and stored in tables while the information in the NoSQL DBs is either unstructured or semi-structured. However, data migration between the relational and the NosQL

DBs is possible. This thesis proposes a methodology for the data migration from MySQL to CouchDB that allows us to move the same data from MySQL into CouchDB. The data is not changed after migration. CouchDB stores and displays information in document form.

The key contributions of this paper are:

- Introducing data migration architecture between the RDBMS and the NoSQL database.
- Developing a DMT (data migration tool).
- Proposing a data migration methodology

The advantages of this tool are multipurpose. It simplifies and automates the process of data migration. At the same time it opens new doors for future development and improvement of this approach.

The rest of the thesis is organized as follows: The next chapter defines the problem of data migration from relational to NoSQL systems followed by a literature review of related work. Chapter 4 presents the architecture and methodology proposed for data migration from the relational MySQL DB to the NoSQL CouchDB. Chapter 5 introduces the conference system developed and used for this research and the reason why it was selected as the prototype. Chapter 6 presents implementation details of this research and Chapter 7 evaluates the thesis research. The experiment and results are covered in Chapter 8 and finally Chapter 9 summarizes the conclusion and future works.

## CHAPTER 2

### PROBLEM DEFINITION

#### **2.1 Introduction to Data Migration from Relational SQL to NoSQL databases**

Relational and NoSql DBs use two entirely different technologies. One of the main differences is the data model of the two systems. The relational databases keep information in tables and join them with the help of primary and foreign keys. They are usually highly normalized and owned by a schema. The NoSQL DB on the other hand, stores information differently e.g. the graph database stores data in graph structures whereas the CouchDB stores data in documents. The stored information in the NoSQL DBs is schema free and highly denormalized. Due to the contradictory nature of the two systems it is a challenge to migrate data from the relational to the NoSQL DBs or vice versa.

There are several relational and NoSQL DBs currently in use. This thesis focuses migrating from the MySQL DB to the NoSQL CouchDB document based database. The rationale for choosing MySQL is that it meets all the requirements of the modern relational databases and easy to manage.

The four main categories of the NoSQL DBs consist of Key-Value, Column Family, Document Databases and Graph DBs. The CouchDB is a document oriented database management system that stores data in uniquely named documents. The document is stored as a Java Script Object Notation (JSON) object with key-value pair and supports semi-structured data that offers some advantage towards the data migration problem. The key-value approach in the CouchDB documents allows for building a database out of the same components that a relational database has internally. This approach allows more control and flexibility over database design.

This also means that in the document-oriented DBs no schema update is required when the data management requirement changes.

### 2.2 Description

The way the RDBMS stores data is very different from how it is stored in the NoSQL DBs due to the specific demands. The data stored in relational databases is highly structured and normalized in most environments whereas the data in NoSQL is mostly unstructured. This difference of data structure helps meet the specific demands of these two systems. The NoSQL DBs are scalable with high availability due to the simpler data model but do not guarantee data consistency at all times. On the other hand the relational SQL systems are not easily scalable with high availability due to the complex data model but guarantees data consistency.

The information in the RDBMS is stored in tables and the tables are interconnected with each other through Primary Key (PK) and Foreign Key (FK). This interconnection is facilitated by the SQL using various types of joins. Figure 2-1 below presents a top level view of how tables are stored in the RDBMS.

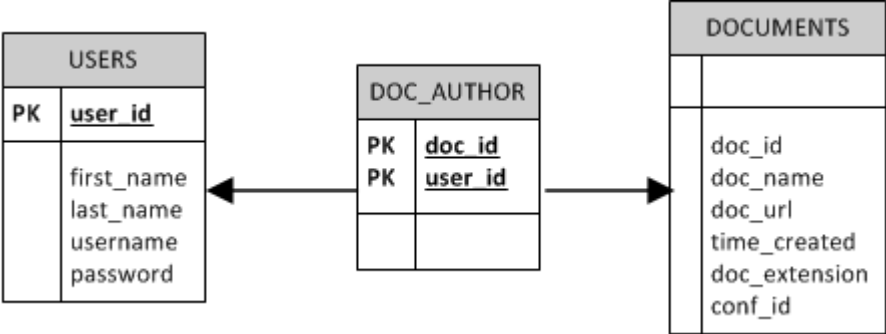


Figure 2 - 1: Data storage in RDBMS

As mentioned earlier the data is stored in the form of the JSON documents inside the NoSQL CouchDB. Figure 2-2 below presents how the data is stored in CouchDB in documents.

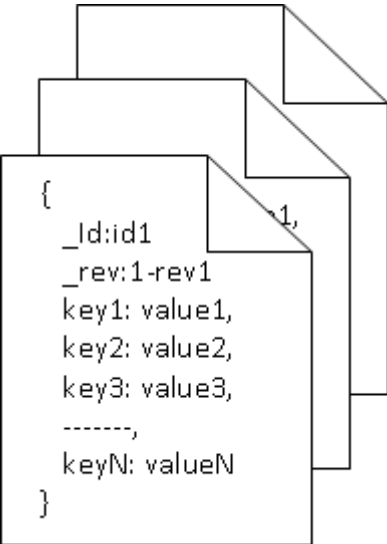


Figure 2 - 2: Data storage in NoSQL

The problem of data migration from the RDBMS to the NoSQL DBs becomes more complex due to entirely different data models and data storage techniques of these two systems. The migrated data in the NoSQL DBs should be able to handle the same operations performed in the RDBMS with the same results. This problem of data migration is exemplified in figure 2-3 below.



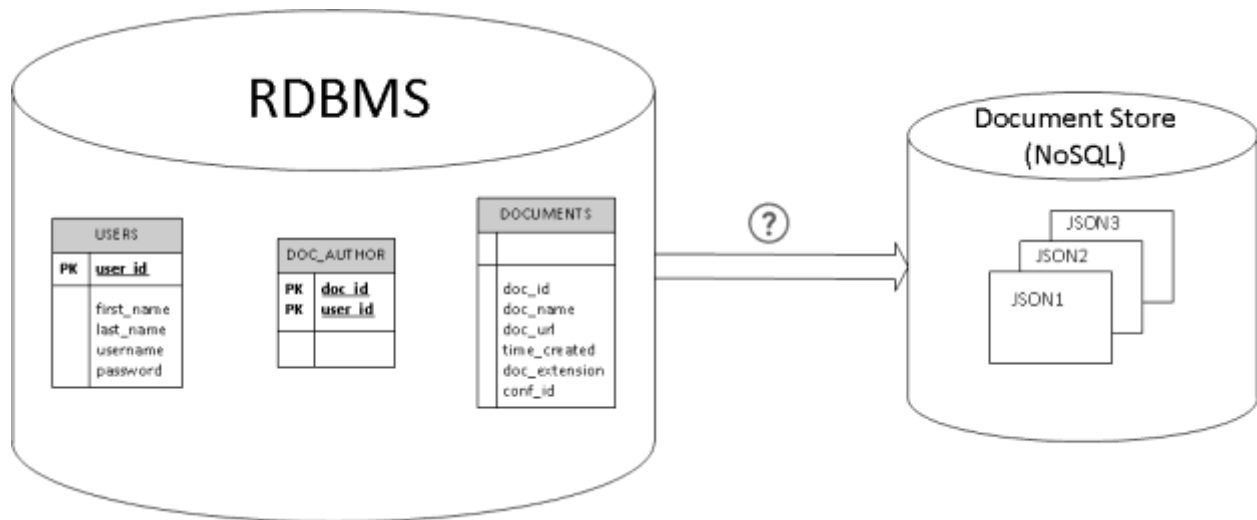


Figure 2 - 3: How data migrate from RDBMS to NoSQL

The problem of data migration is not trivial due to the complexity of data storage structures. It requires migrating data from the relational tables to the NoSQL documents. The challenge is how to handle RDBMS normalization and convert the structure to highly de-normalized and semi-structured or un-structured NoSQL.

## 2.3 Research Goal

### 2.3.1 Goal one: Migration

**G1:** The main goal is to provide a mechanism and to develop a tool to migrate data from a relational MySQL database to a NoSQL CouchDB using a common language such as PHP.

### 2.3.2. Goal two: Identical Functionalities

**G2:** The second goal is to ensure that identical operations can be performed against both databases and get identical results.

### **2.3.3 Goal three: Performance**

**G3:** The third goal is to compare the performance of various operations performed in identical conditions in the two systems.

### **2.3.4 Goal four: Code Comparison**

**G4:** The fourth goal of this research is to determine the complexity of the code written for performing the same operations in RDBMS and NoSQL databases.

## CHAPTER 3

### LITERATURE REVIEW

Knowledge of the internal structures of the RDBMS and NoSQL systems is very important to achieve the goal G1 of this thesis. Internal structure refers to the techniques for how the data is stored in a database. The information about the structure of the database, database objects and the data model form the foundation for understanding data flow in a system. The RDBMS systems revolve around the relational model presented by E.F.Codd [15]. The literature review introduces us with the various concepts presented over the years by E.F.Codd and others. The concepts of normalization [15] and ACID properties plays the role of backbone in RDBMS systems. These concepts are used to develop the basic strategy of data migration from RDBMS to other systems.

The NoSQL databases have different advantages based on their data model structure. There are various types of NoSQL databases and each one of them stores data differently. This thesis focuses on migrating data to the NoSQL CouchDB database. The CouchDB stores information in JSON documents [42]. Information about data storage in the NoSQL databases in general and the CouchDB in particular is necessary to devise a firm strategy to achieve the goal G1. The literature review in the following sections helps us to get as much information as possible to contribute towards the goal G1.

The literature review also looks into the possibilities of finding related information for performance comparison for the same operations to meet the goal G3 mentioned earlier. Similarly it also tries to find related work to meet the goal G4 for code complexity.

This research aims at comparing traditional relational databases with NoSQL DBs. The research also focuses on providing a solution to migrate data from RDBMS to NoSQL DBs. This chapter discusses the relational and NoSQL models to give a better understanding of the

differences in both approaches and thus enabling a better comparison and suggesting a migration method.

### **3.1 Relational Models**

Storing and accessing data has always been challenging. It is essential to organize stored information in such a manner that makes the task of managing and accessing data easy. The future performance of a system heavily depends on how the system was initially designed. Organization and modeling of data plays an important role in how easy or difficult it is to manage the data.

In the late 1960s the Hierarchical data model was used to store information in the form of tree structures. A brief discussion about this model is available in [21, 65]. IBM's Information Management System (IMS) is an example of hierarchic system [21]. The Network data model was another model used along with hierarchical model.

In 1970 E.F. Codd presented the idea of the relational model [15]. The relational model stressed the idea of organizing data in the form of two-dimensional tables called "relations". A relation consists of domain set know as attributes or column having tuples with data for each domain. In the other words a relation is a 'set' of rows or tuples. The idea of presenting the relational model was multipurpose. The relational model was designed to address the following issues as discussed in [15]:

1. Hide from users how data is organized in a machine
2. Protect users activities and applications programs from internal data changes and growing data types
3. Remove data inconsistency

The relational model was aimed at storing information in databases using relations (tables). This means software that allows storing, accessing, and modifying information stored in a computer system (Server). However, this model is used by other types of software too such as the symbol table which is a data structure used by a compiler or interpreter [62].

An analysis of the relational model reveals that there are certain components of a relational model. They are identified as relation, entity, domain, attribute, tuple, and attribute value. These integrated components define the data structure in a relational model.

Figure 3-1 helps in understanding the relational model and its components.

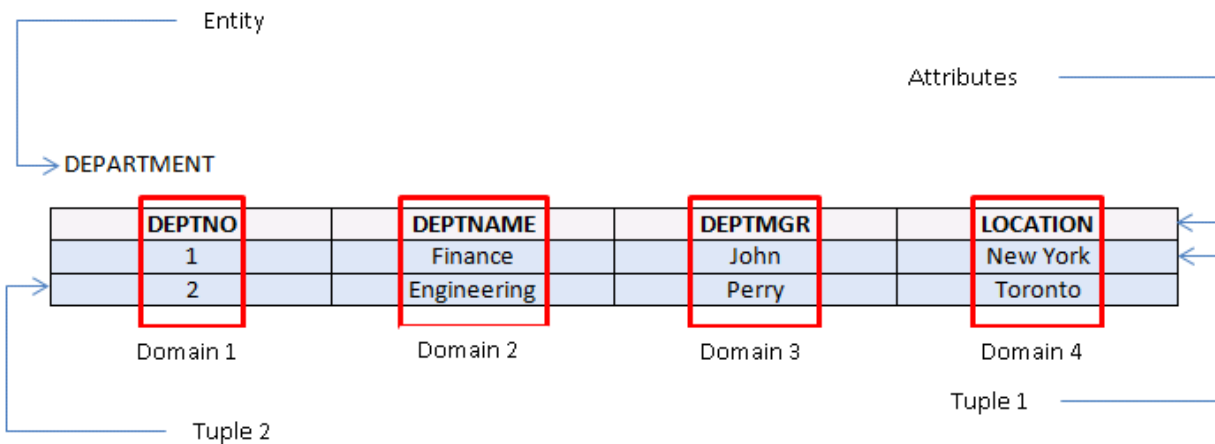


Figure 3 - 1: Illustrates Data Model Structure

The domain in the relational model refers to a set of all allowable values for a specific attribute of a particular type. The table below explains relational model terminologies in addition to the domain.

Table 3-1 below is listing the relational model terminologies with concise explanations.

Relational Model Terminology	Explanation
Relation	Table in a database
Domain	Type of column in a table
Attribute	Column of a table
Attribute value	Column value
Tuple	Row of a table
Entity	Name of a table
Degree	Number of columns in a table
Cardinality	Number of rows in a table

Table 3 - 1: Relational Model terminologies and explanation

Using the definitions as explained in Table 3-1 the degree of relation in Figure 3-1 above is 4 and the cardinality is 2.

E. F. Codd also specified the properties of the relation when the relational model idea was presented. The properties as specified in [15, 16] can be explained as follows:

1. Each row in a relation (table) represents a tuple.
2. Order of rows is not important.
3. No duplicate rows are part of a relation i.e. all rows are distinct.
4. The order of columns (attribute) is not important [16].
5. All table values are atomic i.e. “nondecomposable by the database management system)”

[16].

In 1985, E. F. Codd published a list of 12 rules that concisely defines a truly relational database management system [54].

### 3.1.1 Primary Key Concept

The relational model states that duplicate rows are not allowed in a table to avoid ambiguity [16]. Primary Key (PK) may be defined as a column with unique values that uniquely identifies each record in a table. A primary key consists of a single column or a combination of multiple columns. The Primary key ensures that there is no redundant data. An example of PK is the 'DEPTNO' attribute of the 'DEPT' table in Figure 3.1.

### 3.1.2 Foreign Key

It is common in a relational database that a relation is required to reference its own values or values in another relation. "A foreign key value represents a reference to the tuple containing the matching candidate key value (the referenced tuple)" [21].

Figure 3-2 below gives a pictorial view for foreign key referencing. The attributes 'user\_id', and 'role\_id' of table 'USER\_ROLE' are the foreign keys and reference 'user\_id' of table 'USERS and 'role\_id' of table 'ROLES respectively.

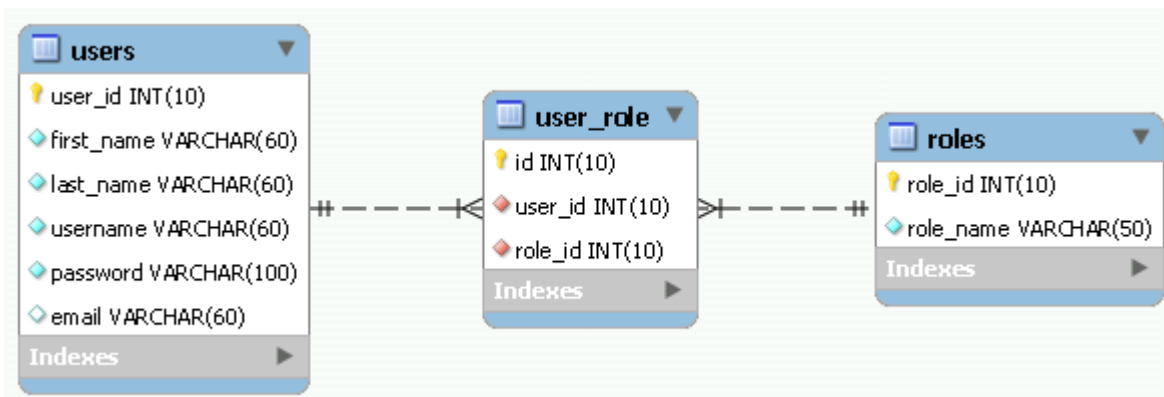


Figure 3 - 2: An illustration of Foreign Key

### **3.1.3 Normalization in the relational model**

Codd initially described the problem of complex domains [15] that can be decomposed into independent sub-domains but related with each other through PKs and FKs. The process was referred to as normalization. This process ensures that data is isolated for each sub-domain and each sub-domain holds data that only relates to that domain, however, all of them are related to each other through foreign keys. The process of decomposing domains into sub-domains implies that a table is divided into more tables which in turn makes the database operations relatively simple and eradicates data redundancy. Codd further discussed the process of normalization in his succeeding paper [17, 18] and introduced three types of normal forms [18].

More research continued with the problem of normalization and to what degree to normalize. This work resulted in more types of normal forms or modification in the existing work [24–26, 41, 43, 45]. However, this paper discusses the first three normal forms known as First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF) as most of the RDBMS meet the requirements of first three normal forms whereas 4NF and 5NF are more for academic purposes.

#### **3.1.3.1 First Normal Form (1NF)**

First normal form (1NF) is the first step in the normalization of data and it was developed as a three step process by E. F. Codd. This normal form requires that all values in a column of a table are atomic. A table is said to be in 1NF if and only if each attribute of the table is atomic.



### **3.1.3.2 Second Normal Form (2NF)**

Again, introduced by E. F. Codd, second normal form requires that the relation must be in 1NF and any non-key column is fully dependent on the entire primary key (a primary key could be a composite key).

### **3.1.3.3 Third Normal Form (3NF)**

Third normal form states that a table is in 3NF if the table is in 2NF and that the non-primary key columns of the table are only dependent on the primary key and do not depend on each other.

### **3.1.4 Process of Normalization**

The normalization process played a great role in the success of RDBMS. The process of normalization decomposes the information into smaller relations and establishes meaningful relationships between them. The normalization process reduces the data redundancy and offers more flexibility to the structure. Codd [13] initially presented the idea of normalization.

Figure 3-3 below represents a simple and clear process of normalization as specified by [1].

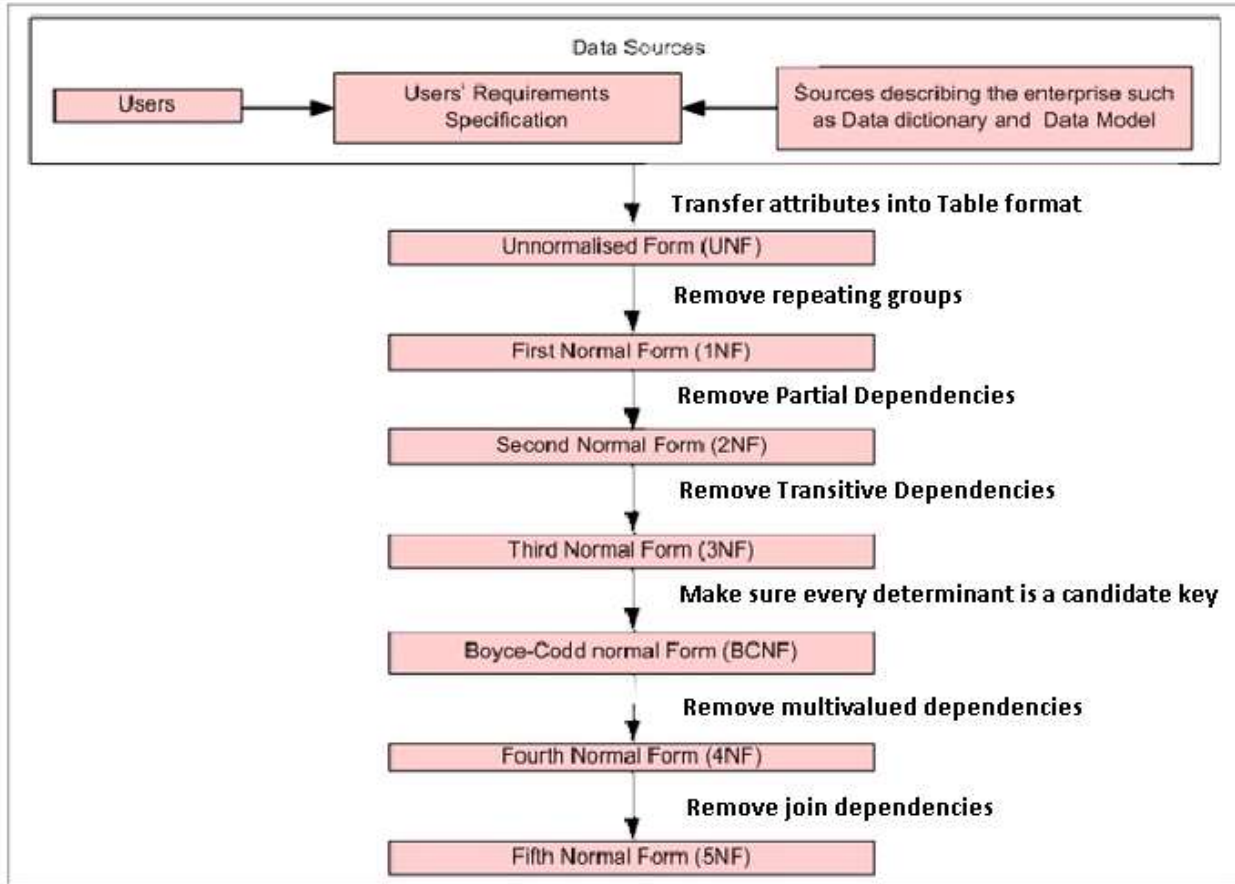


Figure 3 - 3: Possible process flow of normalization [1]

Although normalization is a most widely used process, it is not necessarily an ideal process in each situation. It has been observed that normalization plays important role in Online Transaction Processing (OLTP) systems, however, when it comes to decision making Online Analytical Processing (OLAP) systems such as data warehouses then the fully normalized structure is not helpful.

### 3.1.5 Data integrity and reliability

All RDBMS systems guarantee data consistency. Data consistency is very important as wrong or corrupted data could be detrimental for any business. The RDBMS attains the data

consistency through the Atomicity, Consistency, Isolation, and Durability (ACID) [21] properties. The acronym ACID refers to the four properties in the context of DB transaction processing. They are defined as follows [21]:

#### **3.1.5.1 Atomicity**

“Transaction are atomic (all or nothing)” [21]. A transaction completes as a whole unit. Either it completes successfully or rolls back without making any change. In other words, if any part of a transaction fails then the whole transaction fails without making any changes.

#### **3.1.5.2 Consistency**

Transactions guarantee database consistency. Transactions achieve consistency by keeping the database consistent after a transaction completes successfully.

#### **3.1.5.3 Isolation**

Each running transaction is independent of the other concurrent transactions and they do not see the changes made by another incomplete transaction.

#### **3.1.5.4 Durability**

When a transaction completes the changes in the database remains the same even in case of disaster, errors, or failures.

Keeping the data consistent in a distributed environment is more challenging than on a standalone database. Similarly it is easy to scale the RDBMS on a single server through vertical scalability but when it comes to scale over multiple servers in a distributed environment then the situation becomes more difficult. Nonetheless, modern database vendors have invested lots of

resources for providing highly scalable systems that are also high performance and highly available. Oracle addresses the scalability challenges for both structured (relational) and unstructured data (document, images, multimedia, XML) capable of dealing with Petabytes of data [34]. Microsoft also elaborates how they addressed the issue of high scalability and performance using VMware vSphere™ 4 [60].

The introduction of the web has brought revolutionary changes towards data storage techniques. Scalability and high availability are preferred over data consistency. The nature of web based data is mostly distributed and the RDBMS system is not considered an ideal candidate for huge amounts of the distributed data. According to David A. Grier "If web was considered to be a database, it would fall well outside of Codd's framework" [31]. It further states that the development of the web was based on Codd's ideas and this is obvious from the databases that support the Web. Structural relationships introduced by E.F. Codd, however formed the foundation of information organization [31].

### 3.2 Structured Query Language (SQL)

The relational model has many advantages that made it popular and dominant as soon as it was introduced in the IT industry. In order to access the database an interface was required and that interface was a language starting from sublanguage [15] and passed through several phases to become the modern Structured Query Language (SQL).

Structured Query Language (SQL) is the language used to interact with the RDBMS databases and the object-relational DBs. Most Graphical User Interfaces (GUI) use SQL behind the scene. Any action performed through the GUI runs SQL commands performing certain tasks on behalf of the user. SQL is very much simple English language instructions in a structured fashion.

There are various forms of SQL adopted in the industry. The IBM databases use SQL. Oracle Corporation uses a proprietary extension of SQL in the form of the Procedural Language/Structure Query Language (PL/SQL). Microsoft uses Transact SQL to interact with SQL Server. Other popular databases that use SQL are MySQL, Paradox, and Postgress. However, all these different flavors are based on the industry standard ANSI SQL.

Dr. E. F. Codd presented the relational data model during his research at the IBM laboratory [15] and SQL is closely knit with the emergence of relational databases. SQL has a history behind its reputation. It is a cornerstone of any DBMS system [8]. SQL was first developed by Donald D. Chamberlin and Raymond F. Boyce during 1970s [11]. Initially it was called Structured English Query Language (SEQUEL) and designed to retrieve and manipulate information in the IBM's prototype relational database system, the System R [59]. The name of the language was changed later to SQL because another company already had registered proprietary rights for the acronym "SEQUEL".

Very quickly SQL attained the role of the de facto language for RDBMS as major vendors designed their systems with SQL as the standard language. The popularity of SQL raised the need for a standard to be followed by all. In 1980s The American National Standards Institute (ANSI) and the International Standards Organization (ISO) worked together and developed the first version of the SQL standard under the name of SQL-86 (SQL1). Updating and revision of the SQL standard has been a continuing process since then and the latest version available is ISO/IEC 9075-11:2011 [37].

Despite that the ANSI and ISO standard has been in place for a long time now, database vendors still do not fully follow these standards. They have their own versions of SQL extensions to support their systems. One of the reasons for non-conformity is that the database vendors keep introducing new features and to achieve their desired goals they keep expanding SQL dialects thus resulting in non-standard SQL [32]. Oracle introduced the PL/SQL and Microsoft has Transact SQL, however, all of them still support the basic and major commands of SQL.

Many vendors realized the importance of SQL and embraced it in their products thus increasing the popularity of SQL. The propagation of SQL and the rate of growth of its implementation forced the American National Standards Institute (ANSI) to develop a standard for SQL. The International Standards Organization also recognized the demand of SQL and together with ANSI they released standards for SQL in 1986, 1989, 1992, 1999, 2003 and ISO/IEC 9075-11:2011 [37]. The rationale for a standard was to provide a common platform for future developments and easy migration to third-party applications without changing the SQL code. The standard also helps reducing dependency on the specific vendor. Even after the SQL standard was introduced, several flavors of SQL continue to persist. This was mainly due the

user community demands of a specific database vendor to meet certain capabilities before the birth of SQL standard [38].

Oracle was the first company to introduce the RDBMS system using SQL. The pattern was followed by databases such as DB2, INFORMIX, UNIFI and more. As a result SQL became the standard language for RDBMS systems.

There is a strong relationship between RDBMS and SQL. As discussed earlier Dr. E. F. Codd introduced the relational model of data that suites most forms of data storage structures. Dr. E. F. Codd also described how to interact with a relational database using a language. This language allows data manipulation in a relational model [50]. SQL is a similar language that is equivalent to SQUARE language in functionality but targets those users who are more comfortable with an English like language than mathematical notation for data retrieval [11].

The main reasons motivating the emergence of SQL were the high cost for software development and to enable a common user to interact with databases. Developing software is based on a life cycle that not only involves the initial development phase but it also requires constant modification and maintenance of the software to stay abreast of ever changing requirements. Also most software is used by users who have no or limited knowledge about computers and software. It was a challenge to simplify the explanation to improve the understanding about software and computers for such users. The evolution of Structured Query Language was mainly based on these two critical problems [11].

The RDBMS systems are blamed for slow performance as the number of joins increases. Another argument against them is that they do not map well with complex data programming structures such as XML due to their hierarchical structure. However, the benefits of RDBMS

system are simply taken for granted as the relational technology made it possible for organizations to support their information system on low cost hardware [31].

The RDBMS systems have been in use as the main data storage technology since its inception and have been implemented widely in most commercial and industrial sectors. The popularity and usefulness of the RDBMS remained firm until the introduction of the cloud computing idea. The scenario has changed for the RDBMS since cloud computing started penetrating the industry. This put a high demand on working with really huge amounts of data with a painless scalability option. The RDBMS systems are required to provide better scalability when dealing with large data. However, following ACID rules is a big hurdle for the RDBMS systems to achieve this task. The RDBMS uses a locking mechanism to ensure data consistency but this causes scalability to be a problem in a distributed environment. The web based services further deteriorates this situation as it makes it harder to achieve consistency, availability, and scalability at the same time. This concept is well defined by Dr. Brewer's CAP theorem [61].

Security becomes another issue with the advent of the internet. Web based data requires more security and privacy than traditional application data. New laws were introduced to protect personal information and all organizations are required to follow data protection legislation since 2004 [40]. Basically software vendors are required to exchange personal information as encrypted data [21].



### 3.3 NoSQL databases

Over the years relational database management systems (RDBMS) have been the leading technology in IT world. The RDBMS has dominated the industry since its inception by E. F. Codd presented in 1970 [15]. Several different approaches have been in practice besides the relational concept. Hierarchical and Network data models were introduced in 1960 and implemented by “Integrated Data Store of Honeywell (network model) and Information Management System (IMS) by IBM (Hierarchical Model)” [48]. Being dominant the RDBMS systems embraced and merged other prevailing ideas. Object-based data model was merged with RDBMS to form an object-relational model.

This approach of using relational systems in all scenarios was questioned with the introduction of the web. The Relational data model works well with traditional applications where data is not massive and distributed. Despite being on top for several years the capability of RDBMS for processing and handling large amount of distributed data remains in question. However, handling and processing huge amounts of distributed data is almost mandatory for huge IT companies such as Google, Amazon, Twitter, LinkedIn and Facebook. In short the RDBMS systems are not a suitable candidate for processing large amounts of data where data is distributed over a network in a cluster of servers or over a grid in different geographical locations. A new approach under the flag of NoSQL was introduced to mitigate some of the problems not handled elegantly by RDBMS.

The first time the term NoSQL was used was in 1998 when Carl Strozzi [51] used the term NoSQL for his open source relational database "Strozzi NoSql". Structured Query Language (SQL) was not used to access and manipulate data [51]. Instead, the Application Programming

Interface (API) was used with various plugins and libraries or RESTful API with HTTP protocol. This is the reason why they are called “NoSQL” databases.

The NoSQL DB is a database management system that is entirely different from the RDBMS in many ways. The primary reasons for using a NoSQL DBs are their flexibility, speed, scalability, and high availability. For these reasons NoSQL is making its space in the industry, however, it is still not mature enough and lacks ‘standards’. “For any of the new databases, you may have Pig, Hive, SPARQL, Mongo Query Language, Cypher, or others. These languages have little in common.” [64]. The NoSQL system meets the requirements of corporations who deal with terabytes of data. The data is mostly stored in a distributed environment and does not need schemas or join operations to retrieve data. There is no concept of Primary or Foreign keys in NoSQL since it does not store data in tables and has no constraints. Data in the NoSQL CouchDB is stored in the form of documents. Each document is similar to a row in a table and groups of documents represent a logical table. It uses the advantage of horizontal scaling to improve performance by facilitating the addition of new machines in the system and distributing the additional load equally.

The transaction consistency in the RDBMS DBs is acquired through the ACID properties. The NoSQL databases on the other hand mostly rely on a different concept known as BASE. BASE stands for Basically Available, Soft State, and Eventually Consistent. The BASE concept is defined as follows:

- **Basically Available:** there may be faults but not a fault of the whole system.
- **Soft state:** copies of data items may be inconsistent

- **Eventually consistent:** all the nodes will be consistent when no more updates take place for a long duration allowing all updates to propagate and bring the system eventually to a consistent state.

The BASE design ensures availability but at the cost of inconsistent data. The BASE approach is helpful in designing more scalable database systems, thus allowing more options to add required hardware easily when data grows.

Table 3-2 below represents the main differences with a brief explanation in tabular form as defined in [10, 39, 69].

NoSQL Feature	Description
Structured/Semi-structured	The data in NoSQL is mostly unstructured or semi-structured.
Schema-less	Data is not stored in any schema. Instead data is stored in records and each record has various fields which may change from record to record.
BASE properties	RDBMS systems are mostly ACID compliant whereas NoSQL DBs support BASE properties.
Scalability	Horizontal scalability is the capability of the system to accept the addition or removal of machines/servers and integrate them in a single unit.
Elasticity	Elasticity is the ability of the system to add new hardware without any downtime or interruptions in the services.
Sharding	Shard means a small part of a whole, so the process of sharding partitions large database in smaller and faster chunks across multiple servers. The idea behind sharding is to split data among

	multiple machines while ensuring that the data is always accessed from the correct place.
Asynchronous replication	A technique for replicating data between databases (or file systems) where the primary storage (the system being replicated) does not wait for the data to have been recorded on the secondary or remote storage (duplicate system) before accepting new writes at the primary storage. Asynchronous Replication has the advantage of speed, at the cost of increased risk of data loss due to communication or duplicate system failure.

Table 3 - 2: NoSQL distinguishing properties

Since the introduction of the NoSQL idea numerous databases have been designed depending on the requirements. These databases have been grouped in different categories. Next section discusses and compares current NoSQL designs in use by the industry.

The category of NoSQL depends on how the data is stored. Currently NoSQL DBs are organized into four major categories [52] as follows:

### 3.3.1 Key-Value Stores

The Key Value Stores allow applications to store data in a schema-free environment as a typical Hash table. Hash tables contain a value for a particular key. The key can be system generated or defined by the programmer. The data consists of a key as a string and real data which forms the value part of the pair. The data type of the stored data depends on the programming language and it can be any data type supported by the language. Examples of the

data types are string, integer, array or object. Data stored in such a fashion alleviates the need of a fixed data model and structured data. Amazon was the first to implement key value structure to resolve their data availability and scalability issues using the Dynamo storage system [23]. Key-value stores such as Dynamo are useful in environments with Distributed Hash tables and caching to improve high read performance and "consistent hashing to partition its key space across its replicas and to ensure uniform load distribution" [23]. The key-value structure has been implemented by but not limited to Oracle Berkeley DB [53], Redis [58], MongoDB [57], memcached [33], and Kyoto Cabinet [36]. Figure 3-4 below represents an example of a key-value structure in the CouchDB Futon API. The identifiers for the documents are created as meaningful names rather than system generated strings.

Key ▲	Value
"garysobers" ID: garysobers	{first_name: "Gary", last_name: "Sobers", username: "Gary", password: "mysql", email: "gary@abc.com", role_name: "Student"}
"instructor" ID: instructor	{first_name: "instructor", last_name: "instructor", username: "instructor", password: "62b5d4e4e0d028d8a336607a5bc9d75c", email: "instructor@usask.ca", role_name: "instructor"}
"lynnlee" ID: lynnlee	{first_name: "Lynn", last_name: "Lee", username: "Lynn Lee", password: "81c3b080dad537de7e10e0987a4bf52e", email: "Lynn@usask.ca", role_name: "student"}
"marker" ID: marker	{first_name: "marker", last_name: "marker", username: "marker", password: "62b5d4e4e0d028d8a336607a5bc9d75c", email: "marker@usask.ca", role_name: "instructor"}
"markwaugh" ID: markwaugh	{first_name: "Mar", last_name: "Waugh", username: "MW", password: "mysql", email: "mark@abc.com", role_name: "Student"}

Showing 1-5 of 5 rows ← Previous Page | Rows per page: 10 | Next Page →

Figure 3 - 4: Key-Value document structure in CouchDB

### 3.3.2 Columnar Databases:

The data is stored in columns instead of rows in a Columnar Database. It stores it in a key-value pair but the key is two dimensional. In this type of structure a column key and a row key are required to access a stored value. A timestamp can also be added as part of the key as practiced in Google's Big Table [13]. This type of model is suitable for the handling of huge amounts of distributed data storage as implemented by Google's Big Table. Facebook's messenger services also use Hbase to support billions of messages per day [5].

In this design, a row-by-row approach keeps all the information together in a single entity. On the other hand a column-by-column approach stores together all information about an attribute.

To understand the data structure of a Columnar Database, there are a few important concepts to understand.. These concepts include column family, super column and column. They are defined as follows:

#### 3.3.2.1 Column

A Column in columnar database is simply a tuple with key-value pair. This is the smallest unit as a data container. Google's Big Table also includes a timestamp [13]. An example of Column in JSON notation is as follows [70]:

```
{ // this is a column
  name: "phoneNumber",
  value: "1234567890",
  timestamp: "250920131413"
}
```

### 3.3.2.2 SuperColumn

SuperColumn is a tuple/row consisting of a name and a value. Timestamp is not part of the SuperColumn like the regular Column tuple [27]. The value part is a map containing any number of columns and keyed by the column's name.

A SuperColumn is like a catalogue or collection of other columns to group together multiple columns. Grouping multiple columns using SuperColumn denormalizes the entire row into one SuperColumn [3]. Data stored separately in a column family as a row can also be grouped within a SuperColumn family as a SuperColumn. This strategy improves the performance for the lookup of such data using the value instead of a row key [3]. A simple example of SuperColumn is as follows [70]:

```
clinicAddress: { // this is a super column
    street: "543 xyz",
    city: "Saskatoon",
    zip: "A1B 2C3"
}
```

Regular columns and SuperColumns are both key-value based structures. But the main difference is that the value of a regular column is a "string" whereas the SuperColumn value is a map of columns. Also SuperColumn doesn't have a timestamp component.

### 3.3.2.3 Column Family

Column Families define how the data is stored on disk. Columns or super columns can be part of a column family.

Column keys are grouped into sets called "column families". A Column Family can be considered the counterpart to a table in a relational database. Columns of certain data types are defined in a relational table and then the data is stored in the form of rows/records inserted by the applications. Each row/record contains the same fixed columns.

A Column Family in a columnar database is a collection of rows that contain any number of columns, in a sparse fashion allowing different collection of columns for each row. The column family name is used as prefix for all the column members of a column family. For example, employee:salary and employee:name are both members of the employee column family. The column family can be static or dynamic. In a static Column Family the column metadata is pre-defined for each column whereas a dynamic Column Family allows the application to supply column names to store data.

The columnar data structure looks similar to that of a relational database but the story is different under the hood. In a relational model a row is stored as one record whereas in a columnar structure data is stored by column so data from individual columns are stored together. The order of the data stored in columns is maintained so all parts of a logical row are stored at their respective position in the individual column. This means that for instance if the first name of a person is stored at the 5th position in 'first\_name' column then the corresponding last name will be stored at the 5th position in 'last\_name' position maintaining the integrity of data. The example below explains the concept of column family [70]:

```
clinicAddress= { // this is a ColumnFamily
    northClinic: { // this is the key to this Row inside the CF
        // more required column in this row
        street: "543 xyz",
```



```

    city: "Saskatoon",
    zip: "A1B 2C3"
}

southClinic: { // this is the key to another row in the CF

    // more required in this row

    clinicName: "southclinic",

    clinicType: "children hospital",

    street: "543 xyz",

    city: "Saskatoon",

    zip: "A1B 2C3"

}
}

```

The space allocation as well as the definition of the column is not required unlike the RDBMS structure. The Columnar Database allows adding columns without any burden on the system. A column may be a part of one row and not the other. Similarly a row may have one or more columns. This data structure makes the data retrieval very fast because all the data for a particular attribute is available under a single column. Moreover all columns do not need to be searched if the data is required from only one column. The Columnar Database is also fast for aggregation tasks such as counting the number of employees in a department. Figure 3-5 below illustrates the data structure inside a columnar database for column families and groupings of like column data as mentioned in [55].

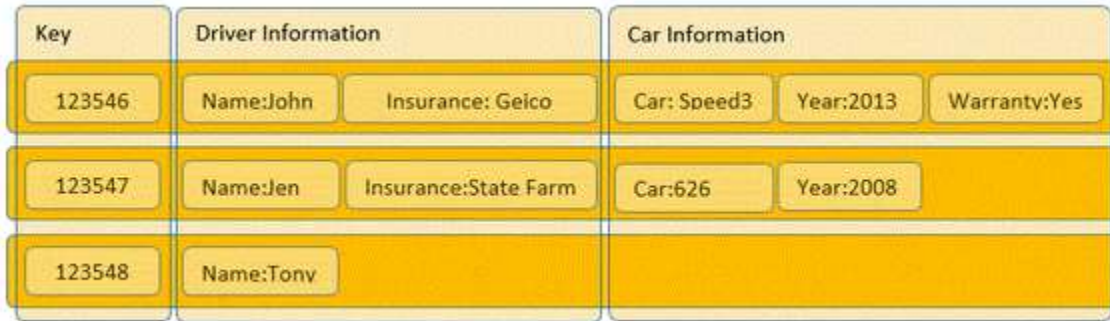


Figure 3 - 5: Columnar database storage structure [55]

The figure above shows Keys with corresponding values but they can store more complex data. This idea is projected in figure 3-6 below comparing the Columnar DB data model with the RDBMS model [55].

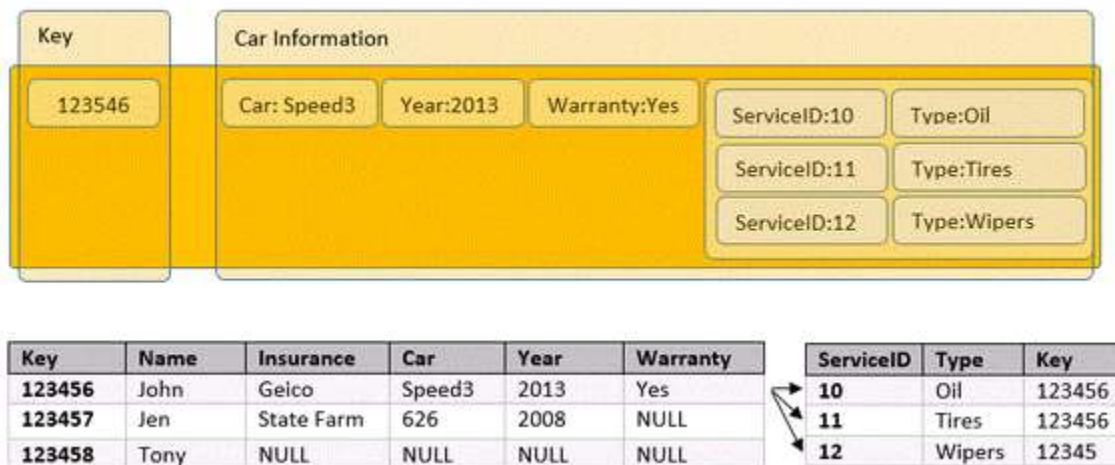


Figure 3 - 6: Same data representation stored in RDBMS and columnar structure [55]

### 3.3.3 Document Store (DS)

The Document Stores allow storing the data in the form of documents. In the RDBMS the data is stored in relational tables and it uses join operations. But the DS consists of several documents each of which is independent of each other and does not belong to any schema. The

fact that all documents are independent of each other makes the programmer's life easy as it relieves them from data dependency and integrity issues. Each document can contain the same or different fields so it eliminates the need for the field value of NULL. If a field is expecting a null value then it is not a part of the document.

All the documents are distinguished using a unique identifier (UID) that is assigned to each document when it is created. These identifiers can be system generated or defined by the programmers. These UIDs works like a PK in RDBMS and ensures the uniqueness of the documents within the database.

As the name suggests Document Stores consist of documents. This model is very useful for horizontal scalability and allows the addition of cheap commodity servers or resources from the cloud as the database grows. The document stores are similar to key-value stores and takes advantage of hashed tables. The key-value pairs are stored in distributed hash tables (DHT) in hash buckets. An index is created on the key-value pairs in these buckets for search purposes. The hash value is a unique number generated from the data. The algorithm for creating hash values distributes the key-value pairs evenly among hash buckets [50]. This means that if the DHT has 4 hash buckets and 20 key/value pairs are stored then each hash bucket will have 5 key/value pairs.

This model makes the parallelization very easy with the addition of more hash buckets when more servers are added due to data growth.

The Document Stores are useful in Web-based applications with data distributed in a network or over a grid. Figure 3-7 [44] below shows how a relation saved in RDBMS is stored in a DS.

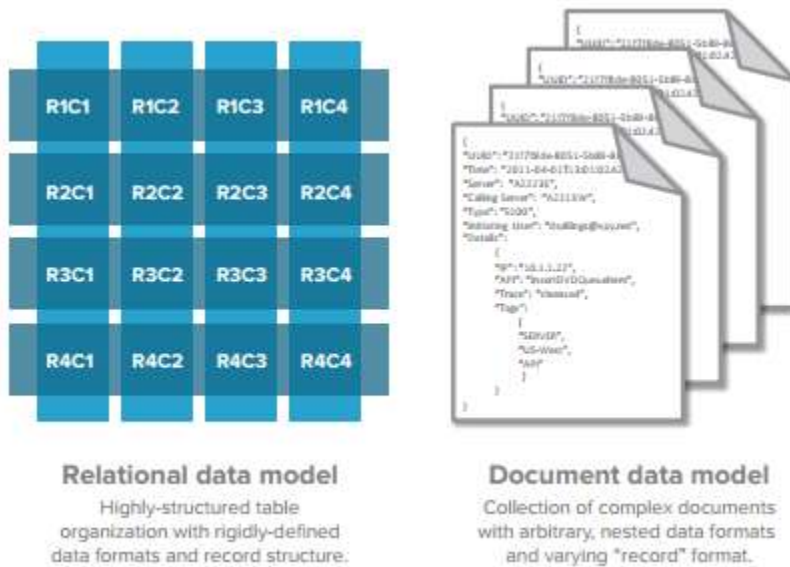


Figure 3 - 7: Documents stored document store [44]

A clearer picture showing the document contents looks like figure 3-8 below [44].

```
{
  "ID": 1,
  "ERR": "Out of Memory",
  "TIME": "2004-09-16T23:59:58.75",
  "DC": "NYC",
  "NUM": "212-223-2332"
}
{
  "ID": 2,
  "ERR": "ECC Error",
  "TIME": "2004-09-16T23:59:59.00",
  "DC": "NYC",
  "NUM": "212-223-2332"
}
```

Figure 3 - 8: Document structure in JSON format [44]

The document model above shows two records in de-normalized JSON format. The records are independent of each other and atomic in nature which makes record movement across servers simple without worrying about the movement of partial data.

### **3.3.4 Graph Model:**

The Graph Databases store and manage graph data and the related indexes. The graphs consist of nodes, relationships among nodes and their properties. Instead of storing data in SQL tables, graph models are used to store the data which is easily scalable over many servers. The main feature of a graph database is how it handles relationships compared to a relational model. The Graph DB traverses through huge numbers of edges in a fraction of the time of relational joins due to direct links between nodes [30]. The Graph DBs use the concepts of vertex (node), edge (relationship) and property. The Graph Model stores data with nodes and edges using a Graph algorithm. This model is useful where the data model is recursive. An example of attributes is as follows:

Node: Employee

Property: first name, last name, designation and department

Relationship: Employee part of a department

The graph databases are useful for social networking web sites such as Twitter that uses FlockDB to store graph data about relationships between users, information about who's following whom and more [35]. Other implementations of Graph DBs are Neo4j, AllegroGraph, Graph DB and InfiniteGraph [2].

The basic structure of a graph consists of nodes and relationships. The nodes have properties and the related data is recorded in Nodes. Relationships also have properties and are used to organize nodes. A random graph structure develops when relationships are defined between different nodes. The resulting structure may be tree like, list, map or a complicated entity that can be further transformed into a more complex structure.

The Graph DBs are queried by using a traversal. A traversal navigates a graph beginning with a starting node and accessing the related nodes using a defined algorithm. It finds answers to the query during traversing e.g. "which of my friends own a car" or "which cities have international airports". This concept of traversal, graph, node and relationship is explained in figures 3-9 and 3-10 below [67].

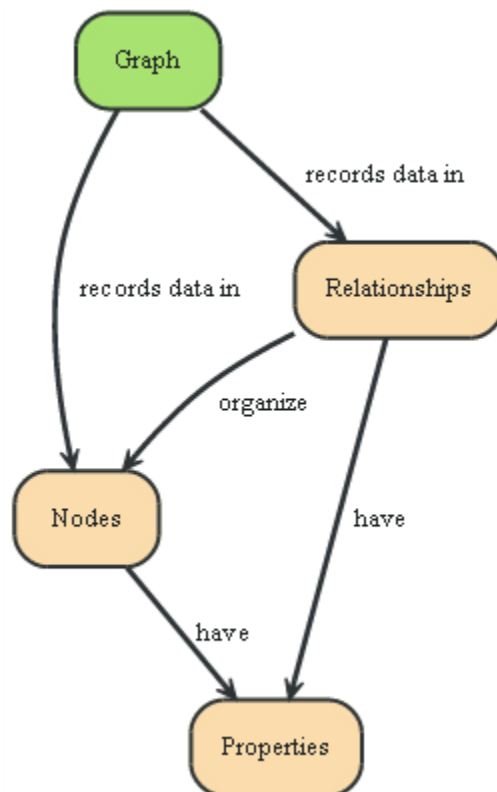


Figure 3 - 9: Graph and basic components [67]

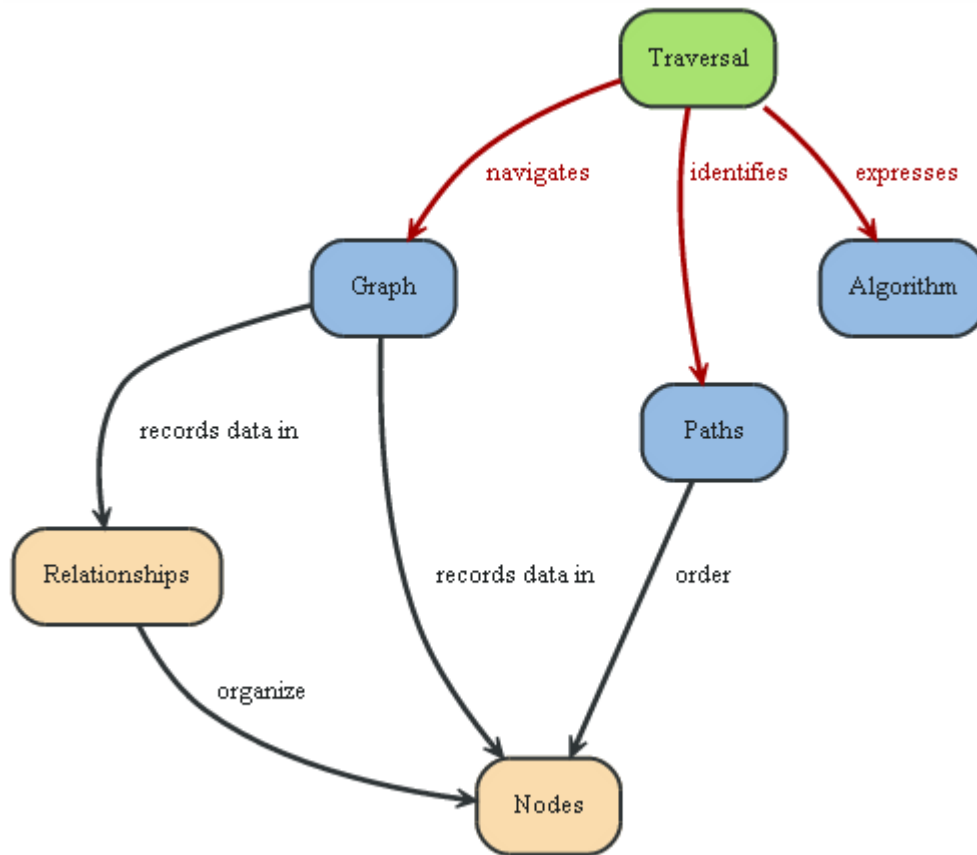


Figure 3 - 10: Graph traversing [67]

It is important that the applications working with graphs retrieve data efficiently. In order to find the existence of a specific node in a graph database, the traversing may perform a sequential access visiting each branch of the graph which could be quite costly. An index can be helpful for fast retrieval of data. To find a particular node or relationship according to a specific property, the use of an index to search for desired information saves time by avoiding unnecessary traversing through the entire graph. The figure 3-11 below shows how an index maps to answer a query [67].

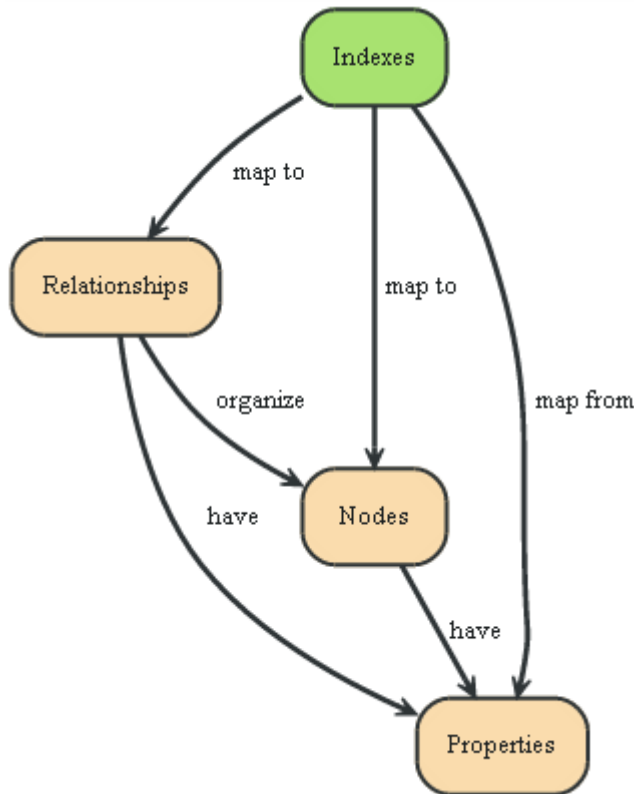


Figure 3 - 11: Usage of index in Graph database [67]

The advantage of the NoSQL databases is in the distributed systems processing terabytes of data and serving hundreds of thousands of users globally. However, processing the huge amount of data in a timely manner is not a trivial achievement. Google implemented a programming technique under the terminology of “MapReduce” which is being widely used by many other well know databases [22].

### 3.3.5 MapReduce

The MapReduce is a programming model that processes and computes huge amounts of resource intensive data in a timely manner in a distributed clustered environment.



Google was the first to introduce the idea of MapReduce [22]. Google has thousands of machines in a clustered environment across the globe and the data is distributed. These machines processes terabytes of unstructured data and compute different kinds of data including but not limited to the most frequent queries at any specific day or number of pages visited per server etc. [46]. Processing small amounts of data was straight forward but with the growth of data in terabytes the distribution of data across several servers became mandatory to complete operations in the desired time. This approach introduced new issues of how to parallelize and distribute data, and how to deal with failures. All these complexities lead to the idea of mapping and reducing data that allows Google to perform simple computations while hiding all the complexities and clumsy details.

The MapReduce model is defined as a Master Worker model [46]. MapReduce uses a library provided by the map-reduce system. The user program spawns a Master process and a few Worker processes at different locations. A Worker process either performs a Map task or a Reduce task but not both. On the other hand a Master handles many tasks. The Master creates new Map tasks and Reduce tasks as selected by the program and also assigns tasks to Worker processes. One Map task is advisable per chunk of the input file(s) but only a few Reduce tasks are recommended because each Map task creates intermediate files for every Reduce task. For this reason too many Reduce tasks may cause the system to be overwhelmed. The Master task also keeps track of the status of each Map and Reduce task. When a Worker process completes the task, it reports to the Master and the Master schedules another task for the Worker process. This process has been represented in figure 3-12 below for better understanding [46].

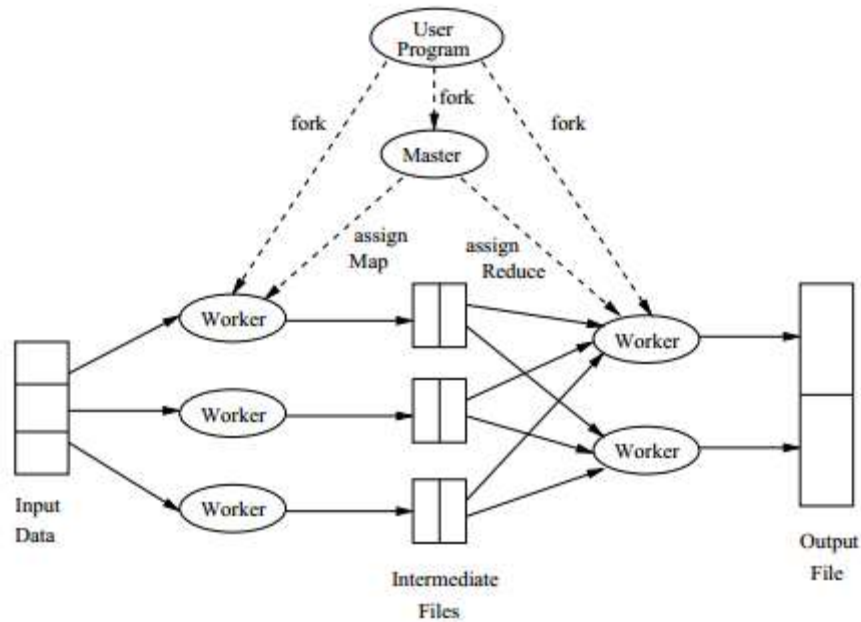


Figure 3 - 12: An overview of MapReduce process [46]

One or more chunks of input file(s) are assigned to each Map task and user written code processes it. The Map function also creates a file on the local disk of the Worker process that executes the Map function. The Master task keeps track of the location and size of each file and assigns the Reduce task to Worker process. A Worker process with an assigned Reduce task also gets all the files to form the input. Finally the user written code is executed by the Reduce task and sends the output to an output file.

In case of failures complete map-reduce jobs must be restarted. Only when the node at which the Master task is executing fails potentially brings the entire process down but other failures are managed by the Master process and the job completes eventually.

### 3.3.6 CAP Theorem

The importance of choosing the RDBMS or the NoSQL databases increases when the implementation is planned in a large distributed system. These systems encounter problems that

smaller systems usually don't face. All RDBMS systems support ACID properties which ensures that data consistency and high availability are achievable. However, implementing horizontal scalability is not an easy task in RDBMS systems. In contrast the NoSQL systems follow BASE properties and deal with system scalability requirements elegantly due to less stringent data consistency requirements. Web based applications usually deal with data that is distributed in nature and grows at lightening speed. Scalability is also required with growing data volumes in a distributed environment in a network or over a grid. Horizontal scalability is easier to achieve with a NoSQL system than a RDBMS system. Implementation in a distributed environment makes it hard to achieve three of the most desired properties namely consistency, availability, and partition tolerance. This concept was first stated in Brewer's Theorem also known as the CAP Theorem. The CAP Theorem states that “It states, that though its desirable to have Consistency, High-Availability and Partition-tolerance in every system, unfortunately no system can achieve all three at the same time.” [29][6]. The individual components of the CAP Theorem are defined as follows:

- **Consistency:** Same data is visible to all clients in a cluster of nodes, even with concurrent updates.
- **Availability:** All database clients are able to access some version of the data in a cluster even if a node in the cluster goes down.
- **Partition tolerance:** The system keeps working even if nodes in a cluster are unable to communicate with each other.

Different systems have different requirements. While consistency in the RDBMS is important, the requirements of the NoSQL DBs are different for different scenarios. The NoSQL systems typically compromise consistency to achieve high availability and latency [4]. Most of the

distributed systems are internet based that guarantees high availability and tolerate network partitions at the cost of eventual consistency. 'Eventual' guarantees that "if update requests stop arriving to the database, then it will eventually reach consistent state" [7]. Some systems require more consistency along with high availability. A better consistency level is possible provided the network availability is good [29]. This can be achieved by setting a threshold value in terms of time for which stale data is acceptable and this can help system designers to define tradeoffs between consistency, availability and partition tolerance. For the systems that need all three properties, a hybrid system of a sharded database in combination with a replicated database has been suggested [68]. The assumption based on the idea that sharded systems are consistent but not available or partition tolerant while the replication systems are eventually consistent but are available and partition tolerant. This means that when a sharded database cannot read data when some nodes crash or are unavailable and loses write access due to a network problem, a fully-replicated eventually consistent database can be used as a fallback to get access to unavailable data from the lost shard.

Figure 3-13 below represents the CAP Theorem visually with a classification of some of the currently available NoSQL DBs [63].

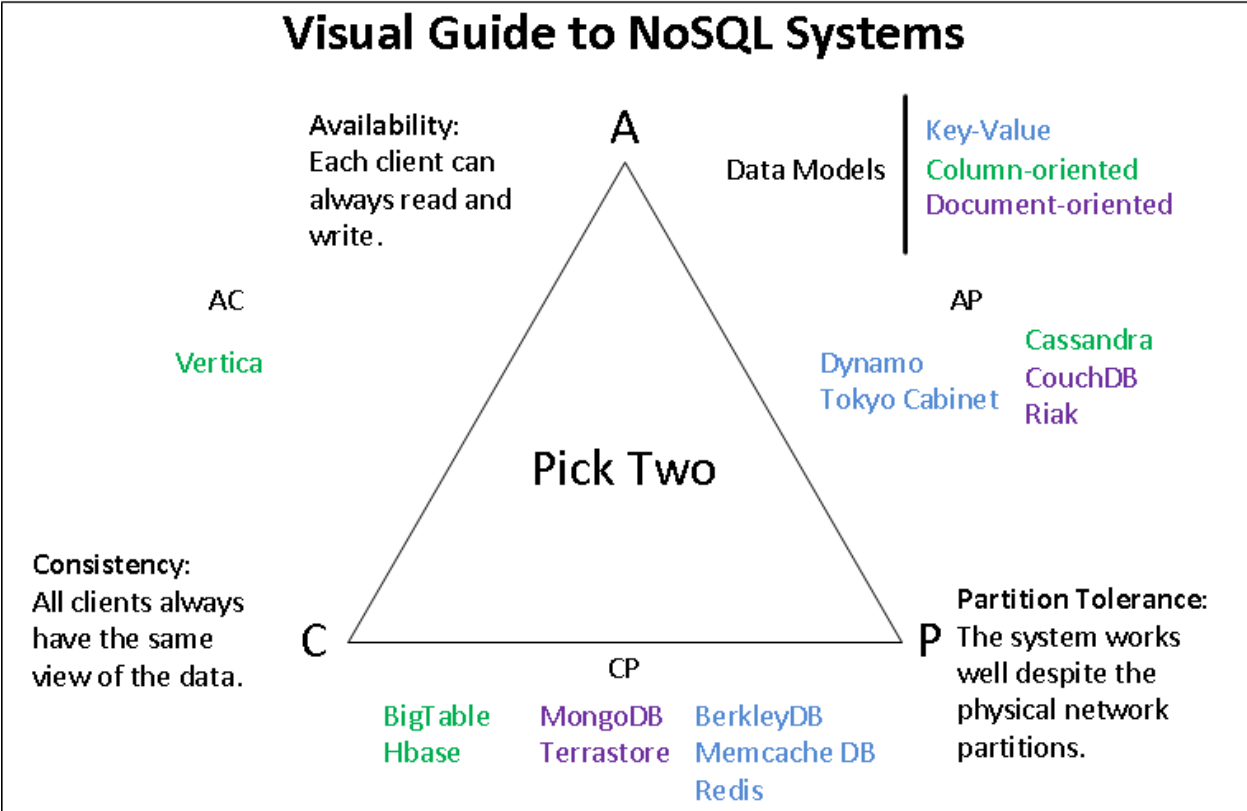


Figure 3 - 13: An illustration of CAP theorem and NoSQL DBs compliance [63]

### 3.4 Data Migration

Over the years most of the data resided in relational databases but the introduction of the NoSQL DBs have changed the paradigm and much of the unstructured data has started going to NoSQL DBs. The RDBMS systems that were working well before the birth of the internet and cloud computing have started feeling the heat of new technologies. New requirements due to the internet and cloud computing has forced the industry to either design their systems based on NoSQL technology to meet the new challenges of high availability and horizontal scaling or migrate data from the standard relational SQL to NoSQL systems.

The data migration in the cloud environment has been discussed by [12] and [66]. The relational cloud system architecture [9] has been explored by [12]. The "Local Conceptual Mapping Database" and the "Centralized Conceptual Mapping" (CCM) components have been suggested as part of the proposed architecture. These components use a data tracker protocol. However, how this tool maps data and if the mapping is done in heterogeneous environments, is unclear. The concept presented in the paper [12] is not substantiated by test results as acknowledged in the 'Conclusion' section. The workflow diagram does not clarify how the data is sent back from the CCM database. The tool proposed in this thesis discusses how data is actually mapped from the RDBMS and the NoSQL and how it is migrated.

The process of migrating from relational to NoSQL databases, however, is not trivial. The main challenge of migrating data from the relational SQL to the NoSQL DBs comes from the entirely different data model structure of the two technologies. The relational databases treat objects as entities and analyze their properties. The entities are mapped to tables that consist of attributes. Relationships between tables is established using the PKs and FKs. There are different types of relationships including one to one, one to many, many to one, and many to many. These relationships are handled in the design of the data model. In particular many to many relationships are resolved through the normalization process decomposing the entities into further smaller units and then establishing relationships among them. Many rules through constraints are also imposed at the same time to avoid data corruption, data duplication and inconsistent data problems. The complexity of the relational model makes it difficult to implement more changes at a later stage. It is not impossible but it makes life difficult.

The NoSQL data model in general is not as complex compared to the relational model and works well in most cases when the data is not normalized. Data duplication and stale data are

used to the advantage of the NoSQL data model to benefit high availability and scalability. Due to the schema free approach of NoSQL it is easy to make changes at any time. However, it requires careful planning and analysis to design an efficient and workable data model before migrating data from a relational to a NoSQL environment.

Not much work has been done in this direction and the existing work is mostly suitable for a particular system that cannot be implemented as is in another system.

The data cleansing during data preparation is suggested by [47] as part of migration process to the NoSQL Cassandra DB. The data cleansing process suggested is supposed to handle business validations but NoSQL DBs have no concept of constraints to handle business rules. Instead business rules are implemented in the application layer. The data cleansing is usually required when data comes from disparate heterogeneous sources and with data types that do not exist in the target location. The purpose of migration is to move exactly the same data from the relational database to the NoSQL DB with the trust that there is no corrupt or redundant data that needs cleansing because it follows the normalization and other relational database rules. However, small data changes may be required to meet the NoSQL data requirements such as uniqueness of the document.

The methodology specified in [56] is specific to the Twitter environment. This method is designed to meet the very specific requirements of the Twitter system and works with the Cassandra DB.

De-normalization is achieved in [56] by duplicating data at multiple places. In contrast the de-normalization in this thesis is achieved by mapping various fields from several relational tables into a single CouchDB document that is used against frequently issued queries to store and

retrieve desired information. This method has the flexibility to generalize and thus is not restricted to one system.

It is interesting that the general impression about the NoSQL technology is that there is no concept of normalization. In fact normalization is a part of the NoSQL DBs too but it is not used frequently. For example, all data could be made part of a single document in CouchDB in a denormalized form but it brings additional problems. These problems forces developers to decompose data into more documents that not only satisfy the application requirements but also improves the performance because multiple processes can write to multiple documents. This avoids the wait time to write to a single document sequentially.

### **3.4.1 Choosing a NoSQL database for the Conference System**

Choosing the NoSQL database for this thesis was challenging. Currently many NoSQL databases have been introduced satisfying various application requirements. A study of comparison among various available NoSQL databases was inevitable to make the right decision.

There are several types of the NoSQL databases currently available. The major types include columnar, key-value, document, and graph databases. Columnar databases are good for decision making systems such as a data warehouse but the Conference Sysytem (CS) developed for this thesis is not decision based. Similarly graph databases are good for following relationships and traversing but this is also not a requirement of the CS system. This leaves us with key-value or document based systems as the database of choice.

The table 3-3 below shows different characteristics of each of the most common databases [19].



Database Name	Developer	Storage Type	Characteristics	Best Use
MongoDB	10gen	Document	<ul style="list-style-type: none"> <li>• Consistency</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	dynamic queries, frequently written, rarely read statistical data
SimpleDB	Amazon	Document		Simple database solutions
ApacheJackrabbit	Apache	Document	<ul style="list-style-type: none"> <li>• Consistency</li> <li>• High Availability</li> <li>• Persistence</li> </ul>	
CouchDB	Apache	Document	<ul style="list-style-type: none"> <li>• High Availability</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	accumulating, occasionally changing data with pre-defined queries
Couchbase	Couchbase	Document	<ul style="list-style-type: none"> <li>• Consistency</li> <li>• High Availability</li> <li>• Persistence</li> </ul>	Session store, user profile store, content store
Cassandra	Apache	Column	<ul style="list-style-type: none"> <li>• High Availability</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	write often, read less

HBase	Apache	Column	<ul style="list-style-type: none"> <li>• Consistency</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	random read write to large database
Riak	Basho Technologies	Key-value	<ul style="list-style-type: none"> <li>• High Availability</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	high availability
Big Table	Google	Column	<ul style="list-style-type: none"> <li>• Consistency</li> <li>• High Availability</li> <li>• Partition Tolerance</li> <li>• Persistence</li> </ul>	designed to scale across hundreds or thousands of machines

Table 3 - 3: Characteristics of various NoSQL databases

Table 3-3 above represents a comparative study of various currently available open source NoSQL database as described in [19]. A close look at the characteristics of various databases reveals that they serve virtually the same purpose in different ways such as high availability, consistency, partition tolerance, and persistence. Scalability is also a common factor in all the NoSQL databases. It is also obvious that all of the NoSQL databases are popular because they are easily scalable without disrupting the online operations. All these characteristics make it a really difficult task to choose the right database system. However the environment in which the NoSQL database will be used and the current relational database structure and usage is relatively helpful in making a decision.

The test bed conference system developed in the relational MySQL database does not anticipate high activity or dynamic queries. The operations are mostly specified with pre-defined queries and more read operations are anticipated than write operations. The CouchDB is document based and also takes advantage of the key-value structure. For this reason, the CouchDB was a good fit for the CS system. The document oriented CouchDB is a good candidate based on the characteristics defined above in table 3-3. The document data stores are an extension of the key-value store taking full advantage of the key-value stores. Data is stored as JSON documents in the document store along with a key. These documents are accessed in the database using the corresponding key. Creating and maintaining documents in the NoSQL CouchDB database is very simple and allows storage of structured data in the form of unstructured or semi-structured data in the NoSQL database.

Another important feature of CouchDB is the Multi-version Concurrency Control (MVCC) System that avoid transaction level locking unlike the MySQL database. The concurrent access to the database in CouchDB is controlled using MVCC [20] instead of a locking mechanism as practiced in the RDBMS systems. The figure 3-14 below illustrates the differences between the MVCC and the relational DBs locking mechanisms.



Figure 3 - 14: MVCC locking mechanism in CouchDB [20]

MVCC simplifies the mechanism of accessing the same document through various sessions. New versions of a document are created in the CouchDB when a change is incorporated and the newer version is saved over the old version resulting in multiversions of the same document.

“How does this offer an improvement over locks? Consider a set of requests wanting to access a document. The first request reads the document. While this is being processed, a second request changes the document. Since the second request includes a completely new version of the document, CouchDB can simply append it to the database without having to wait for the read request to finish.

When a third request wants to read the same document, CouchDB will point it to the new version that has just been written. During this whole process, the first request could still be reading the original version. A read request will always see the most recent snapshot of your database.” [20]

All these characteristics and advantages makes CouchDB the best candidate as the NoSQL database for this thesis.

### 3.5 Conclusion

There are some clear differences between the NoSQL and the RDBMS. Also there are few clear advantages and disadvantages in both systems over the other. Sticking to the ACID properties makes the RDBMS most reliable for data integrity and users are prevented from accessing stale data. But this has created problems for the RDBMS to scale in a distributed environment with data in terabytes or more. The RDBMS could not meet the demands of big companies when it came to scalability and high availability. The data consistency needs locks on the data and scalability requires downtime. Both affect the performance and availability of the system.

On the other hand these problems have been addressed in the NoSQL DBs. Eventual consistency leverages the NoSQL DBs to achieve high scalability and availability by allowing users to access stale data until all the databases in a distributed environment are eventually consistent with the same data across all nodes. The NoSQL DBs take advantages of data sharding and replication to distribute the data at several nodes. The absence of the schema concept in the NoSQL DBs makes it possible to add new nodes for horizontal scalability and all this processing remains seamless to users.

Use of the NoSQL technology is gaining popularity while relational databases have maintained a concrete place in the market for decades. It is not surprising that traditionally data migration is required by organizations to serve various goals. Also data migration happens from various sources to meet the requirements of the organizations. Scaling is a big problem in a relational system when data grows out of the capacity of one machine. To resolve the issue of scalability and high availability in relational systems it is possible to move them to the NoSQL DBs thus a need arises for data migration from relational to the NoSQL DBs. Although some system specific strategies have been adopted to meet the specific requirements to migrate data

and some methodologies have been suggested, yet there is no tool available for data migration. There is a need of a DMT that could be used in most environments and have the flexibility of generalizing to different environments.

Not much scholarly work is available for the topic of this thesis but plenty of information about data models and data structures is helpful to understand how data is stored and accessed in both technologies. This information is useful in converting the RDBMS data model into the NoSQL data model which is the basic requirement for success of the goal G1 of this thesis. The literature review also provides information about different components that can be combined together to support a workable architecture for data migration. The concept of mapping data presented in [12] has been implemented differently in this thesis. Not much information is available currently that covers this important aspect of future demand. However, this fact cannot be ignored easily. This thesis provides an architecture and methodology to migrate data from RDBMS DBs to NoSQL DBs to achieve the goals defined earlier.

## CHAPTER 4

### DATA MIGRATION ARCHITECTURE

This chapter first presents different types of scenarios and features such as tables, relationships between tables and the concept of constraints in the RDBMS as learned from [14, 17, 18, 24, 41, 43]. Most of the available information is about the data models and internal structures of the two systems i.e. RDBMS and NoSQL. This information has been used to define the document structure in the CouchDB to meet the application requirements. This document structure is defined based on the data model in the RDBMS and how information is kept in various database objects such as relational tables. However, not much scholarly information is available about data migration between them. The literature review helped with identifying different components of the proposed architecture to achieve goal G1 which is the primary goal of this thesis.

The main components of the suggested architecture in this thesis are the NoSQL CouchDB, the relational MySQL DB and the interface with the MySQL DB and the CouchDB. The PHP language supports connection to the MySQL DB using a connect function whereas the PHP client libraries are used as discussed in [28] to connect to the CouchDB from the user interface. The connections to the MySQL and the CouchDB databases make it possible to perform further operations towards achieving the goal of migrating data. Also a flat file introduction as a hub between the CouchDB and the MySQL DB allows for manual adjustment and a data cleansing operation if required. As a unit this architecture helps in achieving goal G1 and subsequently allows achieving goals G2, G3, and G4.

The next section in the chapter briefly explains the rationale of choosing the CouchDB with the MySQL DB based on the available information about relational and NoSQL systems. It then describes the steps for migration and the migration architecture with the help of a figure.

## **4.1 Types and scenarios in relational database**

Relational databases are composed of entities essentially represented as tables. These tables have certain constraints in the form of the primary and foreign keys. A table can only have one primary key but there is no restriction on the number of foreign keys. A table may have none, one or more than one foreign key which indicates that a table with foreign key references a primary key table. The primary and foreign keys also links tables with each other and establish relationships. There are different possible scenarios in relational database as defined below.

### **4.1.1 Table with primary key**

There are tables that are referenced by other tables in a relational schema but do not refer to any tables themselves. These are tables that have a primary key but do not have foreign keys. There are two types of primary keys that may exist in a table.

#### **4.1.1.1 Single Column primary key**

A single column primary key consists of a single column in a table. An example of this is the USERS table in the conference system designed and used in this thesis. Figure 4-1 shows the structure of a table with the USER\_ID as the primary key. A key beside the USER\_ID column identifies it as a primary key.



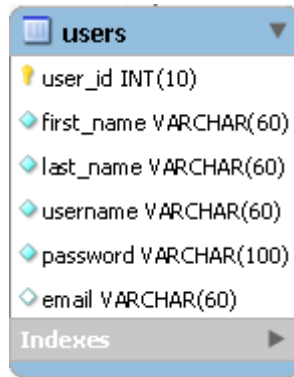


Figure 4 - 1: A table with single column primary key

#### 4.1.1.2 Composite primary key

The primary key that consists of more than one column is called a composite primary key. An example of this is the DOC\_AUTHOR table whose primary key consists of the USER\_ID and the DOC\_ID. The DOC\_AUTHOR table references the USERS and the DOCUMENTS tables respectively. Figure 4-2 below shows the structure of the DOC\_AUTHOR table. A key beside the DOC\_ID and the USER\_ID columns identifies them as the composite primary key.



Figure 4 - 2: Relational table with composite primary key

#### 4.1.2 Tables with a single foreign key

A table can have single or multiple foreign keys. When only one column of a table references another table then it is a table with single foreign key. An example of a table with single foreign

key is the REVIEWED\_DOCUMENTS table with the REVIEWER\_ID column referencing the USERS table. Figure 4-3 below the structure of this table with a diamond beside the REVIEWER\_ID column that identifies it as a foreign key.

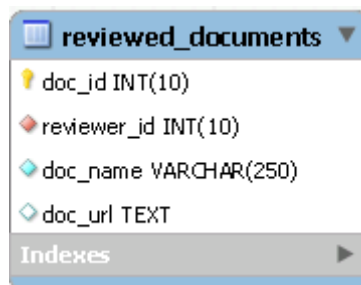


Figure 4 - 3: Table with single foreign key

#### 4.1.3 Tables with multiple foreign keys

It is common in real life that a single table references more than one table using multiple columns. A table with more than one field referencing other tables is said to have multiple foreign keys. An example of a table with multiple foreign keys is the REVIEW\_STATUS table that references the USERS and the DOCUMENTS table through the REVIEWER\_ID and the ORIGINAL\_DOC\_ID columns. A diamond beside these columns in figure 4-4 identifies them as the foreign keys.

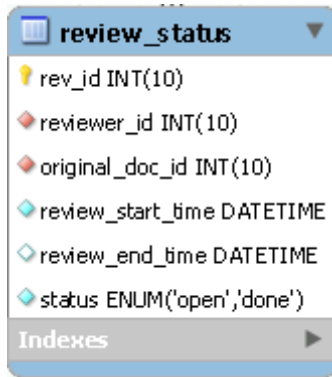


Figure 4 - 4: Table with multiple foreign keys

Other than the above mentioned types there are various types of relationships that exist in relational databases. They are identified below.

#### 4.1.4 One-to-one relationship

In an one-to-one relationship each entry in the first table has one, and only one, counterpart in the other table. This type of relationship is not commonly used in real life scenarios because in this case it is more efficient to combine the information in one table. This type of relationship does not exist in the conference system.

#### 4.1.5 One-to-many relationship

In an one-to-many relationship each entry in the first table corresponds to one or more entries in the second table but each entry in second table corresponds to one, and only one, entry in the first table. The conference system has many relationships of this type. For example a user can have many documents but each document belongs to only one user.

#### **4.1.6 Many-to-many relationship**

In many-to-many relationships each record in both tables corresponds to one or more records in the other table. This type of relationship is usually resolved by normalization by introducing an intermediate table. Both the tables are related to each other through the intermediate table by having one-to-many relationships with the intermediate table. An excellent example of this type of relationship in conferencing system is between the USERS and the DOCUMENTS tables where each user can review more than one document and each document can be reviewed by more than one student. This situation has been handled with the normalized data model.

#### **4.2 Migrating from relational MySQL to NoSQL CouchDB**

The information kept in relational databases is structurally differently from information kept in the NoSQL CouchDB database. Tables are generally the main source of information storage in the RDBMS. However, the same information is kept in the form of documents in the CouchDB [42]. Each row in a table is equivalent to an entire document in the CouchDB.

The CouchDB is a document oriented DB that can be queried and indexed in a MapReduce fashion using JavaScript [42]. It provides a RESTful JSON API that allows the HTTP requests. The views in the CouchDB use the Map function and facilitate consolidated information in a table like structure for all the documents that meet the function's criteria.

Creating the documents in the CouchDB and populating them with data from the RDBMS is possible. This document presents the architecture and the method to convert a table into the CouchDB documents. The concept behind this approach is to create a similar structure for each document in the CouchDB based on a single table. The number of the documents is equal to the number of rows in the table. Each document, however, maintains a unique identification either

by using a system generated id or a manually assigned id. Manually assigned IDs are meaningful while system generated IDs are randomly generated complex strings. All the documents created in the CouchDB based on a single MySQL table contain identical fields that are similar to the source table. The only exception is that the documents in the CouchDB maintain uniqueness using a ‘\_id’ mandatory field. This field is equivalent to a primary key in a table. The CouchDB allows assigning meaningful value to ‘\_id’ field as mentioned above. The other field in the document that is automatically maintained by CouchDB is ‘\_rev’ field. This field maintains the uniqueness of each revision of a specific document for availability purpose.

Figure 4-5 below shows the high level architecture of migrating data from the MySQL tables to the CouchDB documents.

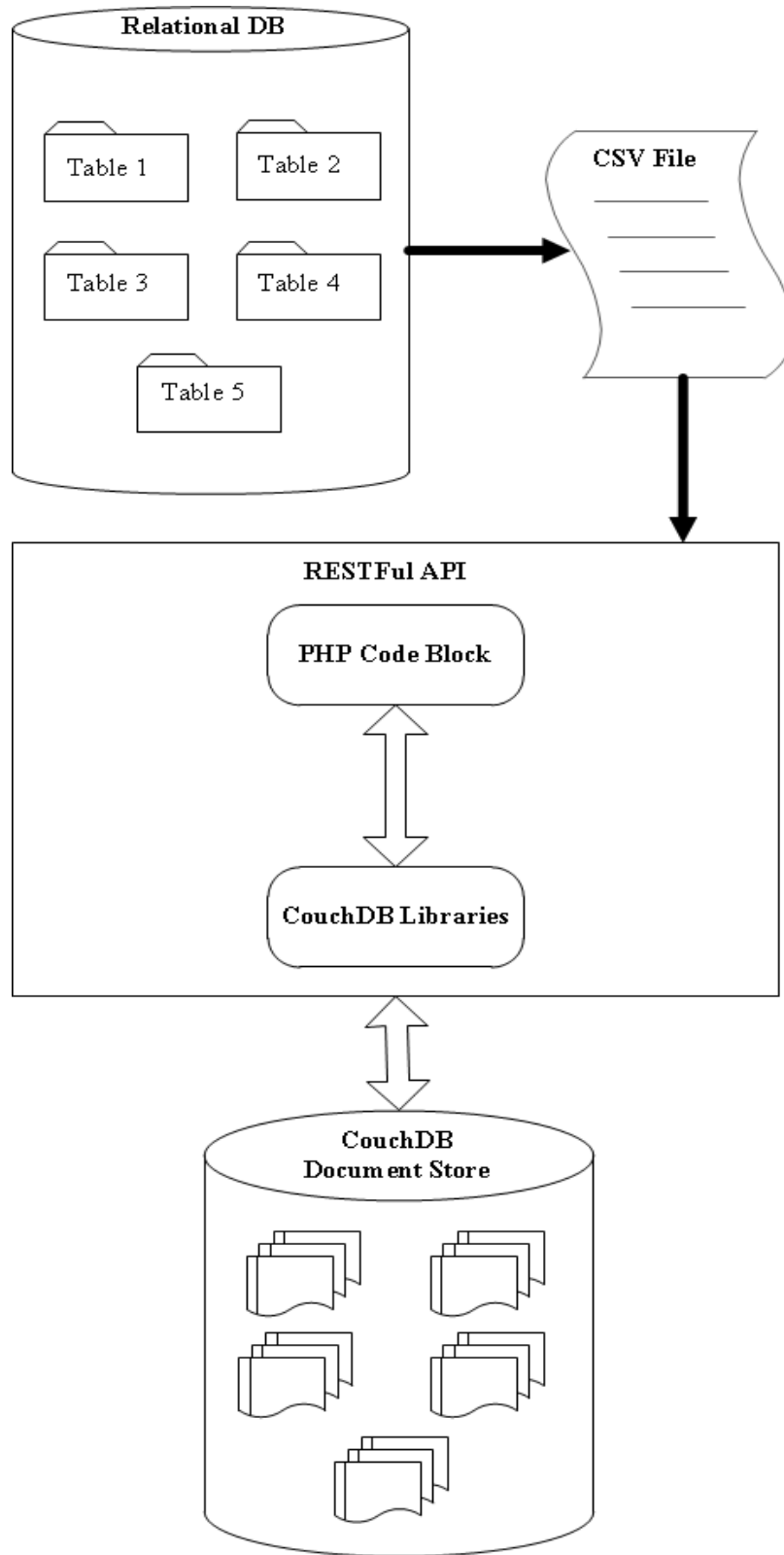


Figure 4 - 5: Data Migration Architecture

According to this architecture several SQL queries are used to extract the information from the MySQL database based on the requirements in the CouchDB. A document in the CouchDB may require information from only one table which is a rare case in real life or it may require information from several tables which is a common scenario. Due to the de-normalized structure of the CouchDB documents all information can be collected in one document. As the CouchDB supports the RESTful API, it allows the tool to communicate simultaneously with the CouchDB, read PHP code block and interact with the file to extract information and convert it to the CouchDB documents. The desired document structure is required in the PHP code block. The tool reads the code and creates the document in the CouchDB with the same structure defined in the code block. It also populates it with the data at the same time. Chapter 6 includes more details about the implementation.

## CHAPTER 5

### TEST BED: CONFERENCE SYSTEM

This section introduces the RDBMS prototype used for the migration procedure, the reasons for choosing a conferencing system and the underlying framework. This chapter also describes the implementation approach and data model for both the RDBMS and the NoSQL DBs. Part of this chapter describes the high level architectural design of both implementations

#### **5.1 Introduction to the Conference System (CS)**

The Conference System is an application through which users can share information. The CS in this thesis primarily makes use of computers and the internet. The CS system may be expanded to use technologies and equipment such as audio, video, TV/monitor screens and many more. The information is shared and managed using a web browser over the internet and all information is stored in a database at the backend. There are two main reasons for choosing the CS system for this thesis.

The first and most important reason is that there was a need in the Computer Science department to develop a CS system. This system is required for use in class and to share documents between students and to perform various tasks such as commenting, rating, and grading documents. This system not only helps the program chair (Instructor or Marker) to organize conferences but also helps authors and reviewers (students) in performing their activities.

The second reason is that this system gives sufficient and rich functionality along with a desired level of data complexity that is adequate to be implemented in both the RDBMS and the NoSQL for fair comparison.



The RDBMS system chosen for this comparison is the MySQL because it has all the desired features and functionalities of relational databases. MySQL is open source and easy to maintain over a long period as compared to rival Oracle or SQL Server databases.

The CouchDB was selected as a good representative of NoSQL databases as described earlier. The CouchDB DB is a document storage system that stores data in the form of individual documents independent of each other. It is interesting to see how a relational model behaves when converted to a document storage system such as Couch DB. The next section discusses the data model for the CS in the RDBMS.

The CS system under discussion meets the core requirements and follows the workflow as defined in the next section in addition to other secondary features.

### **5.1.1 System requirements and workflow**

The CS system works as follows:

1. The instructor posts assignments and the due dates.
2. Students submit assignment within the due dates.
3. The system automatically assigns each assignment among all students for review based on pre-defined criteria by the instructor.
4. Each student's (e.g. student 1) assignment is assigned to two other students (e.g. student 2 and student 3).
5. Student 2 and student 3 review the submitted assignment of Student 1 and give it a rating and makes comments'.
6. Student 1 also reviews the assignments of some other students (e.g. student 5 and student 6) assigned to Student 1 for review.

7. Student 1 does not see the ratings but sees the comments made by the Student 2 and Student 3 on her/his assignment.
8. Now, Student 1 rates the 'comments' of the Student 2 and the Student 3 and submits the final version of the assignment.
9. The marker (TA) awards the grade to the final version of assignment submitted by Student 1 and gives 'comments'.
10. Student1 sees the feedback (comments) and grades about his/her assignment.
11. Also the Student 1 sees evaluations of the reviews s/he gave to Student 5 and Student 6 (i.e. Student 1 sees the comments made by Student 5 and the Student 6 on the comments that s/he (Student 1) made about their assignment). Student 1 also sees the reviews provided by the other (anonymous) reviewers. In addition Student 1 sees the ratings that Students 5 and 6 gave to all reviews they received.

## **5.2 RDBMS Conference System Data Model**

The conference system contains several tables, relationships and constraints. This section discusses the role of each entity to store and retrieve the desired information. The last part of this section contains a diagram that shows the data model of the CS system. The details about the tables of the CS system are included in Appendix A.

Figures 5a-1 and 5b-1 below presents a data model diagram designed using the MySQL data modeling tool for the CS system. This figure not only shows the relationship between the different entities but also the primary and foreign keys and data types. For clarity purpose the figure has been divided into two parts.

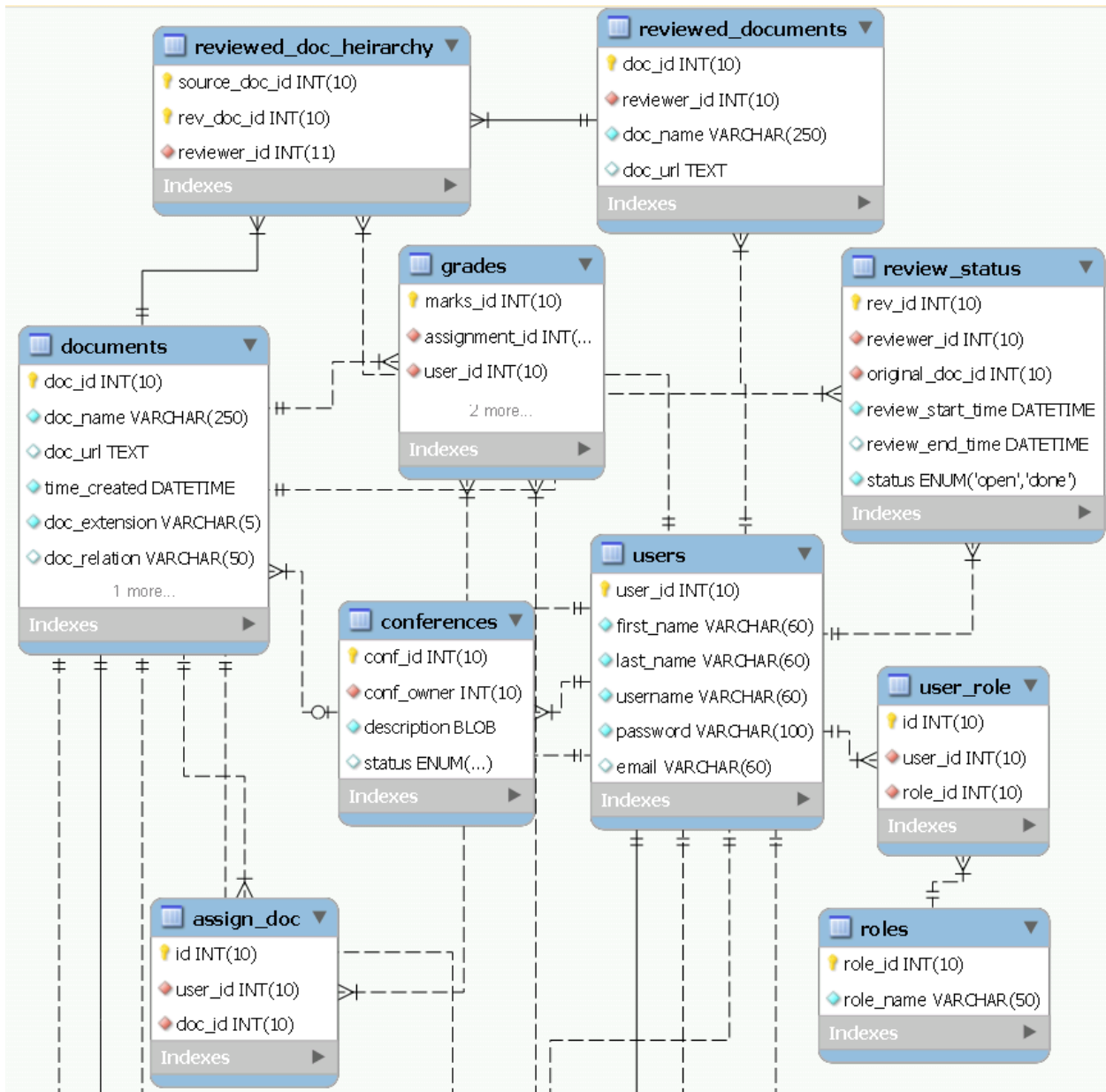
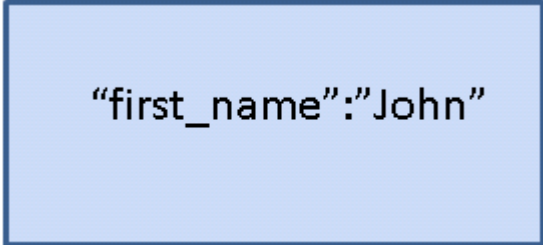


Figure 5a - 1: Conference System Data Model (part 1)



object consisting of named fields. The field values may be either simple data types like strings, numbers, dates or complex data types such as associative maps or ordered lists. Collectively these documents form the database and each document is identified by a unique identifier. The data is stored as a key-value structure in a document. The CouchDB uses the key-value structure to store data in the JSON objects. The data model may have objects with simple key-value structure. The simplest form of a key-value construct is one key with a value. The figure 5-1 below shows the simplest form of this construct.



```
"first_name":"John"
```

Figure 5 - 1: Simplest form of key-value pair

In a more complex key-value structure the value part may consist of objects. An object is an unordered set of name-value pairs and it can be used as the complex value component of a key-value construct. The figure 5-2 below represents the values as objects.

```
"user": {  
  "first_name": "John",  
  "last_name": "Doe",  
  "role": "student"  
}
```

Figure 5 - 2: Object as complex component of key-value structure

The value in a key-value construct may also consist of an array. An array is an ordered list of values that need not be of the same type. Thus a document can be represented with an unbounded nesting of array and object constructs. The Figure 5-3 below represents complex value consisting of an array.

```
{  
  "author_name": "JohDoe",  
  "role": "Student",  
  "document": {"doc_name": "Assignment 1",  
    "doc_url": "https://ashikpc.usask.ca/drs/assignment1-20121217-104438.pdf",  
    "time_created": "2012-12-17 22:44:38"},  
  "reviewers": [  
    {"first_name": "Jessy", "last_name": "Ryder"},  
    {"first_name": "Tim", "last_name": "May"}  
  ]  
}
```

Figure 5 - 3: An array as complex component of key-value structure

### 5.3.1 Views

The views in CouchDB play an important role in the overall data model. The data in the CouchDB is semi-structured that eliminates the need for join operations. The CouchDB offers

the ability to create views using JavaScript for description. A view is itself a document that enables a client to filter and organize the documents according to application requirements. The views are virtual in nature and can be created dynamically without any impact on the database activities or underlying documents. The CouchDB offers two types of views as discussed below.

#### **5.3.1.1 Permanent Views**

These views are created and stored in special documents known as design documents. These views can be accessed using the HTTP GET request. The Uniform Resource Identifier (URI) prefixes `_design` and `_view` helps recognize that the request is for design document or for a view. A design document in CouchDB is a special document that can be accessed using HTTP GET requests and has the prefix `_design` to help CouchDB recognize the document as design document. The first time access to a permanent view may be slow depending on the number of documents in the database while the CouchDB creates the view. The views are not created and updated when a document is saved but instead when they are accessed. All the views in a single design document get updated if any of the views in the design document is queried.

#### **5.3.1.2 Temporary Views**

Temporary views are not stored in the database permanently instead they are executed when required. To execute the temporary view the URI contains the prefix `_temp_view` which tells the CouchDB that the request is for a temporary view. The temporary views are very expensive and therefore not used regularly. They are used to test results and if the results are good then saved as a permanent view for future use.

## CHAPTER 6

### IMPLEMENTATION

This chapter consists of the prototype implementation of the data migration approach presented in Chapter 4. The key components of the implementation are the MySQL and the CouchDB databases, RESTful API, PHP Client libraries for the CouchDB, and the Apache or compatible Web Server.

The implementation of this tool uses SQL to interact with the MySQL database and the PHP scripts to interact with and perform actions in the CouchDB. Data is transferred through an intermediate file in Comma Separated Value (CSV) format.

The figure 6-1 below shows high level implementation architecture of these components. One server is used to implement all components of this research.



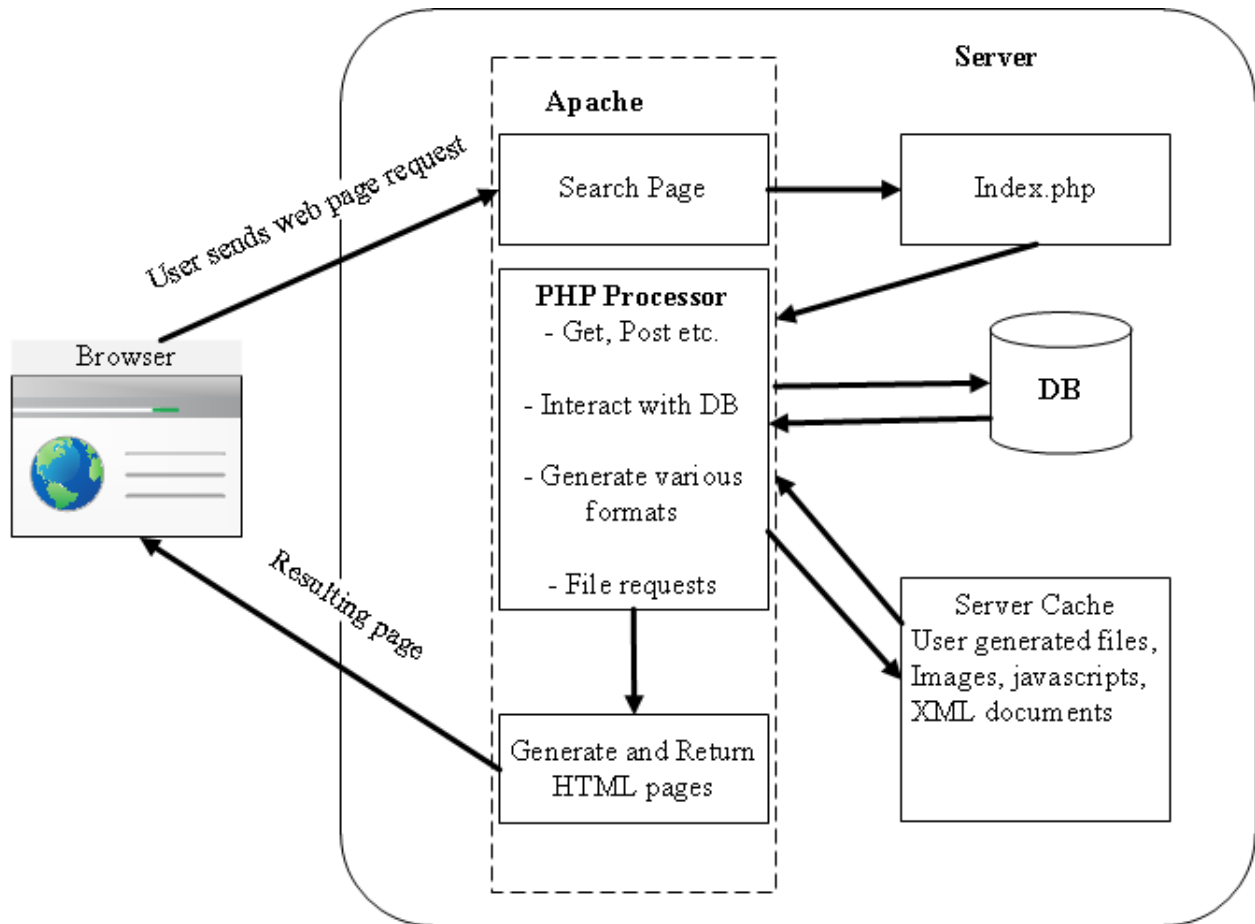


Figure 6 - 1: High level implementation architecture

### 6.1 Data Migration Steps

The process of migrating data consists of steps converting the relational tables to the CouchDB structure defined below.

- Determine the CouchDB document structure i.e. fields required in the document
- Extract and examine the MySQL tables structure i.e. determine existing fields in the tables

- Define the document structure based on the above steps i.e. define the fields required in a particular CouchDB document from one or more MySQL database tables
- Create CSV file containing data from a source table using a script
- Embed the final structure of CouchDB document in the PHP code block. This code is read during the migration operation and a document is created in the CouchDB according to the defined structure.
- Create document and import data in CouchDB using CVS file and CouchDB libraries

Figure 6-2 below shows a simple partial code example of embedded structure in PHP.

```

if ((isset($_POST["user_id"])) && ($_POST["user_id"] == "form1")) {
    $doc = new stdClass();
    $doc->_id = $_POST['doc_id']; // the unique id of the document

    // let's add some other properties
    $doc->first_name = $_POST['first_name'];
    $doc->last_name = $_POST['last_name'];
    $doc->username = $_POST['username'];
    $doc->password = md5($_POST['password']);
    $doc->email = $_POST['email'];
    $doc->role_name = $_POST['role_name'];
    try {
        $client->storeDoc($doc);
    } catch ( Exception $e ) {
        die("Unable to store the document : ".$e->getMessage());
    }
}

```

Figure 6 - 2: PHP code with embedded structure of the CouchDB document

The process is currently manual but can be automated in future. The definitions of the objects are extracted by looking at each object individually and documenting it. These definitions are

obtained either by using a reverse engineering or by just defining the MySQL table inside the database. The process starts with creating a CSV file with data from the source table.

The process of converting the RDBMS table starts after the table definition is collected from the RDBMS. Data in the RDBMS is manipulated using the Structured Query Language (SQL) through customized client software or a module such as MySQL Workbench. The CouchDB allows use of the RESTful HTTP API to communicate with the database and provides the user with access to the information. Representational State Transfer (REST) is an architecture style that makes it possible to access data using web services that are implemented in the Hypertext Transfer Protocol (HTTP). REST was introduced by Roy Fielding in 2000. Everything in a REST base architecture is a resource and these resources are accessed by common interface based HTTP standard methods GET, POST, PUT, and DELETE. A resource can be anything e.g. a document residing on a server in a file system or a table row in a database and these resources are accessed by Uniform Resource Identifier (URI). In the REST architecture the client can access and modify the resource on a server. The server provides and controls the access to the resource. REST supports all common HTTP operations and allows the resources to have different representational styles such as XML, JSON, and HTML etc. A client access the representation by using the URI.

The Extensible Markup Language (XML), HyperText Markup Language (HTML), and JavaScript Object Notation (JSON) are popular formats of data for the RESTful applications. The method in this research uses a JSON representation embedded in a PHP code block which interacts with the CouchDB using the CouchDB libraries for PHP. The structure of the documents is defined in the PHP code block and data is read from the CSV file. Using the CouchDB API, an HTTP request is issued to the CouchDB server using GET, POST, PUT, or

DELETE methods. The data is passed along with the request method and a new document is created in the CouchDB. This method provides precise control over the unique identifier of the new document. It allows assigning a meaningful and unique identifier to the CouchDB documents. The unique identifier `_id` is automatically assigned to a new document, if not defined explicitly.

This utility is flexible and provides the facility to provide a hardcoded value for each key entered through a form. Another method is to read an existing file with exported data using the API form. This operation can potentially be automated in future research.

Documents in CouchDB are created with a key-value structure. The keys in this case are the name for the table's fields and values are the data in the table. The uniqueness of each document is maintained by giving it a meaningful name.

The figure 6-3 below explains how the RDBMS table converts into the CouchDB document. This also shows how table fields are mapped to counterpart keys in a document system and how the same data is assigned to each key as a value. This explanation uses an actual table of the CS system that is converted into a CouchDB document with identical data.

A table 'USERS' with all users' information is populated with 3 rows. This table generates three documents in the CouchDB when converted. Each row in the table is mapped with exactly one document. The fields' name and data are identical. Each document name is unique as explained earlier. The NoSQL DBs have no concept of Primary Key so the PK column is not required in the documents. Each newly created document is assigned a uniquely identified document through the '`_id`' key. The name of the document identifies it and could also be considered equivalent to a table name.

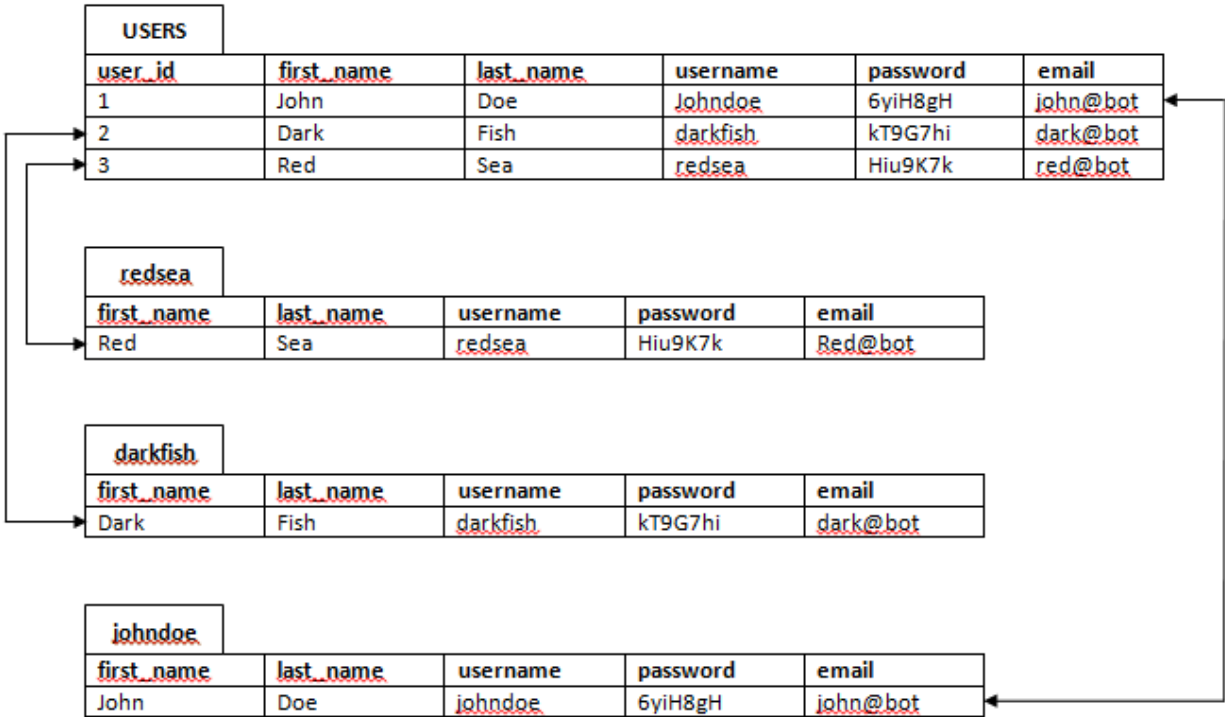


Figure 6 - 3: Mapping RDBMS table with CouchDB document

The first scenario described in Figure 6-3 above is good if the data is being imported from one MySQL table but the CouchDB model is semi-structured and also de-normalized as compared to the MySQL tables. It is common that due to the de-normalized nature of the data structure, the CouchDB documents contain fields for which the data may be coming from multiple tables. When this situation is encountered then the desired field from the MySQL tables should be mapped to the corresponding fields in the CouchDB documents. The rest of the process is identical as if we import the data from a single table. To use this approach the programmer has to be aware of the CouchDB data model and document structure. Consider a document in the CouchDB that is required to contain information about the review process of a submitted document. The CouchDB document structure contains information as shown in figure 6-4 below.

Field
<b>_id</b>
<b>_rev</b>
<input checked="" type="checkbox"/> <b>review_end_time</b>
<input checked="" type="checkbox"/> <b>review_start_time</b>
<input checked="" type="checkbox"/> <b>review_status</b>
<input checked="" type="checkbox"/> <b>reviewed_document_dir</b>
<input checked="" type="checkbox"/> <b>reviewed_document_id</b>
<input checked="" type="checkbox"/> <b>reviewed_document_name</b>
<input checked="" type="checkbox"/> <b>reviewed_document_url</b>
<input checked="" type="checkbox"/> <b>reviewer_name</b>
<input checked="" type="checkbox"/> <b>source_doc_author</b>
<input checked="" type="checkbox"/> <b>source_doc_id</b>
<input checked="" type="checkbox"/> <b>source_doc_name</b>
<input checked="" type="checkbox"/> <b>source_doc_url</b>

Figure 6 - 4: Structure of a CouchDB document

The record that populates the above mentioned CouchDB document is shown in Figure 6-4 in parts for clarity below.

review_end_time	review_start_time	review_status	reviewed_document_dir
2012-12-23 13:11:47	2013:11:18 12:54:57	done	2390

Reviewed_document_id	Reviewed_document_name	Reviewed_document_url	Reviewer_name
8	Using CouchDB	<a href="https://ashikpc.usask.ca..">https://ashikpc.usask.ca..</a>	Lynn Lee

Source_doc_author	Source_doc_id	Source_doc_name	Source_doc_url
Tim May	1	Using CouchDB	<a href="https://ashikpc.usask.ca.">https://ashikpc.usask.ca.</a>

Table 6 - 1: A record in MySQL

Same record looks like Figure 6-5 below in a CouchDB document after data migration.













Field	Value
_id	LynnLee_reviewed_TimMay_assignment1
_rev	1-8f08c5fc5426c01367c27a629746b0d0
 Review_end_time	2012-12-23 13:11:47
 Review_start_time	2013:11:18 12:54:57
 Review_status	done
 Reviewed_document_dir	2390
 Reviewed_document_id	8
 Reviewed_document_name	Using CouchDB
 Reviewed_document_url	http://ashikpc.usask.ca...
 Reviewer_name	Lynn Lee
 Source_doc_author	Tim May
 Source_doc_id	1
 Source_doc_name	Using CouchDB
 Source_doc_url	http://ashikpc.usask.ca...

Figure 6 - 5: MySQL record as CouchDB document after data migration

The information in this document is coming from multiple MySQL tables including USERS, DOCUMENTS, REVIEWED\_DOCUMENTS, and REVIEW\_STATUS. The structures of these tables are represented in figures 6-6, 6-7, 6-8, and 6-9 respectively. To import data in this scenario, proper mapping of table fields with the document fields is essential. Each row in the tables creates one new document in CouchDB with the fields as controlled in the coding.



## USERS

Field	Type	Null	Key	Default	Extra
user_id	int(10) unsigned	NO	PRI	NULL	auto_increment
first_name	varchar(60)	NO		NULL	
last_name	varchar(60)	NO		NULL	
username	varchar(60)	NO	UNI	NULL	
password	varchar(100)	NO		NULL	
email	varchar(60)	YES		NULL	

Figure 6 - 6: Structure of USERS table in MySQL

## DOCUMENTS

Field	Type	Null	Key	Default	Extra
doc_id	int(10) unsigned	NO	PRI	NULL	auto_increment
doc_name	varchar(250)	NO		NULL	
doc_url	text	YES		NULL	
time_created	datetime	NO		NULL	
doc_extension	varchar(5)	NO		pdf	
doc_relation	varchar(50)	YES		NULL	
conf_id	int(10) unsigned	YES	MUL	NULL	

Figure 6 - 7: Structure of DOCUMENTS table in MySQL

## REVIEWED\_DOCUMENTS

Field	Type	Null	Key	Default	Extra
doc_id	int(10) unsigned	NO	PRI	NULL	auto_increment
reviewer_id	int(10) unsigned	NO	MUL	NULL	
doc_name	varchar(250)	NO		NULL	
doc_url	text	YES		NULL	

Figure 6 - 8: Structure of REVIEWED\_DOCUMENTS table in MySQL

## REVIEW\_STATUS

Field	Type	Null	Key	Default	Extra
rev_id	int(10) unsigned	NO	PRI	NULL	auto_increment
reviewer_id	int(10) unsigned	NO	MUL	NULL	
original_doc_id	int(10) unsigned	NO	MUL	NULL	
review_start_time	datetime	NO		NULL	
review_end_time	datetime	YES		NULL	
status	enum('open','done')	NO		open	

Figure 6 - 9: Structure of REVIEW\_STATUS table in MySQL

Figure 6-10 below represents how the MySQL table fields are mapped to the CouchDB document fields during the process of import.

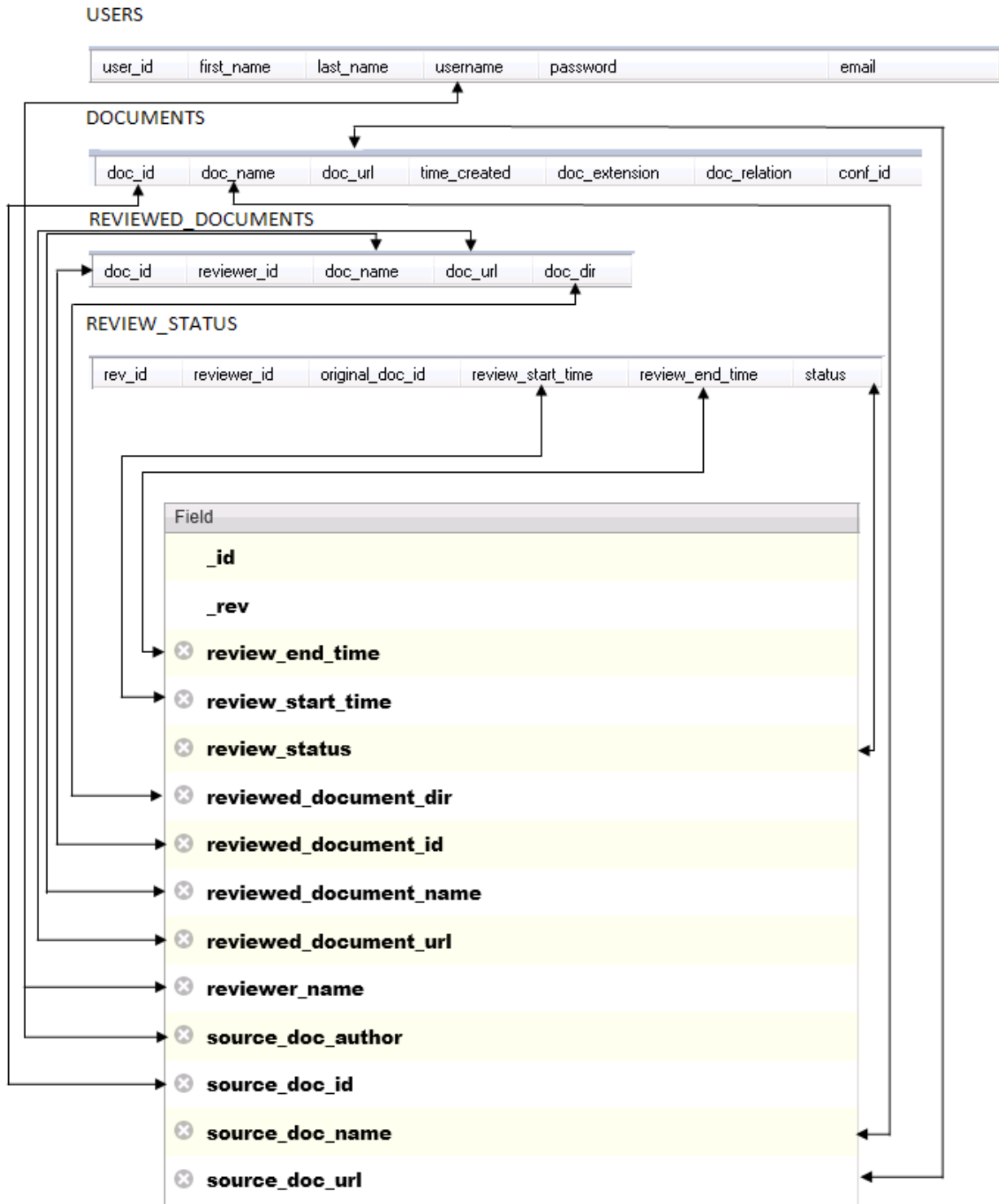


Figure 6 - 10: Mapping multiple table fields in CouchDB document

## CHAPTER 7

### EVALUATION

The main focus of this research is migrating data successfully without any data loss or data corruption and then to use the migrated data to perform identical operations and get identical results. The goal G1 is achieved successfully if the data has been moved from the MySQL to the CouchDB successfully without any data loss or corruption. The goal G2 merely depends on the success of the goal G1 and is achieved successfully if the same operations i.e. SELECT, INSERT, UPDATE, and DELETE that were performed in MySQL DB produce the same results with the migrated data. There is no direct dependency of the goals G3 and G4 on goals G1 and G2 but the success of the first two goals confirms that the code written to achieve the goals G1 and G2 is correct. Once it is confirmed that the code is correct then we can use it to achieve the goal G4 by comparing the codes written for both the MySQL and the CouchDB to perform the same operations. The goal G3 is totally independent of the other goals and depends on other factors too.

#### **7.1 Evaluation Goals**

In short, the evaluation will look into the following four aspects based on the goals.

##### **7.1.1 Has the data been migrated successfully to CouchDB?**

This will be verified by comparing the data in the MySQL DB with CouchDB documents. The data inside each document should match a corresponding record in the MySQL database.

### **7.1.2 Can the same operation be performed in both systems and achieve the same result?**

There are various operations performed in the databases. The four operations are select, insert, update, and delete.

The DMT has been designed to connect to both the MySQL DB and the CouchDB and perform the same operations on the same data. The results of the performed actions will confirm that the operations can be successfully performed on migrated data and returns the same results.

### **7.1.3 What is the speed difference for performing the same operation in both systems?**

The data migration is the main aspect of this thesis but another goal is to compare performance while performing the same operation in both systems. A time estimate for performing SELECT, INSERT, UPDATE, and DELETE operations will be a helpful tool to provide an idea about efficiency of both systems for each operation.

### **7.1.4 How complex is the code for the same operation in both systems?**

Separate scripts will be written for both systems to perform the same operations in both systems. A comparison between the scripts written for MySQL DB with its corresponding script written for CouchDB will be helpful to determine the complexity of the code. Code complexity will be measured by comparing the PHP scripts written for both systems for the same operations. The number of lines of code is the baseline for measuring code complexity and the comparison provides a general idea of how much more difficult it is to write code for a MySQL DB versus a CouchDB.

## 7.2 Experiment Goals

The methods of evaluating the goals for this thesis are outlined in this section. Keeping the goals in view the evaluation will look into the following four experiments.

The first goal was “**G1:** First and main goal is to provide a mechanism and develop a tool to migrate data from a relational database to a NoSQL CouchDB using a common language such as PHP”

This goal relates to the question if the data has been successfully migrated using this tool or not. Below is how we evaluate this question.

### 7.2.1 Has the data migrated successfully to CouchDB?

There are two scenarios under this category.

#### 7.2.1.1 Data migration from single table in multiple CouchDB documents

In this scenario the data comes from only one table and is decomposed into multiple documents when it reaches the CouchDB. The number of records in the table will convert into an equal number of documents. The process of the experiment is shown in the figure below.

Complete knowledge and information about the structure and requirement of the destination system are essential before starting the process. The queries will be executed in the MySQL DB to export data into flat files according to the required structure in the CouchDB. These queries are shown in the figure 7-1 below as Q1, Q2, Q3,..., Qn. Each query in the MySQL database produces results that match the corresponding structure of the desired CouchDB document. These generated files with the required data will be created and will reside on the file server as F1, F2, F3,..., Fn respectively.

In the second part of the process the PHP scripts S1, S2, S3,...,Sn will be written with embedded structure of each required document in the CouchDB. These scripts use all the components as mentioned in the DM architecture in Chapter 4 and will read files F1, F2, F3,..., Fn with the exported data. The data will then be imported into CouchDB through these scripts. A high level concept of this approach is captured in the figure 7-1 below.

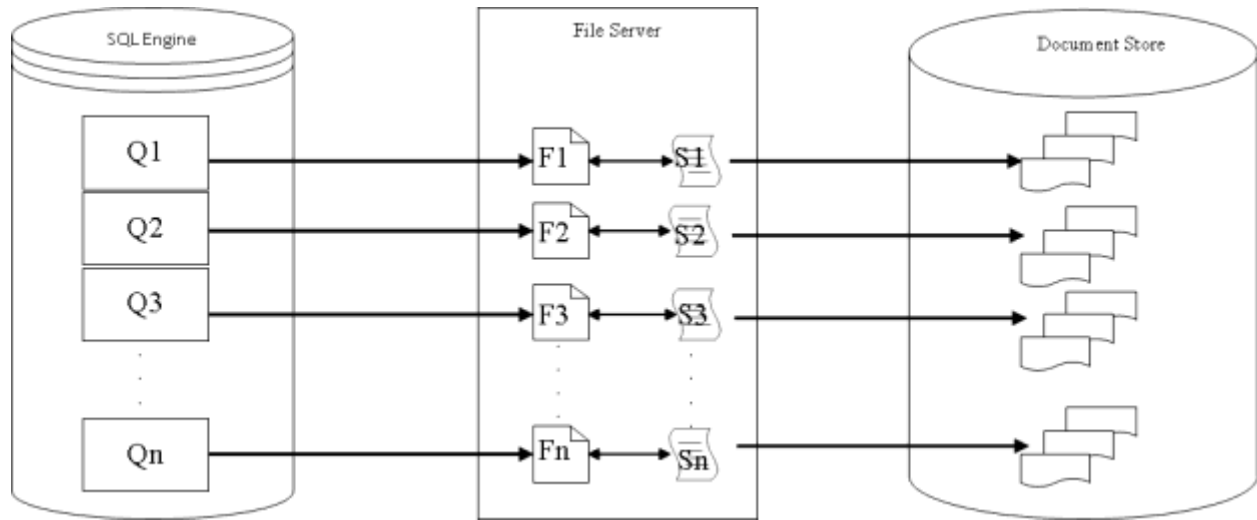


Figure 7 - 1: Data Migration Process flow

Database queries are run in the MySQL DB based on the information and data structure required in the CouchDB. When a query runs it creates a corresponding output file populated with the required data. Each customized PHP script corresponds with only one generated file with the exact structure incorporated in the code block for the purpose of creating documents in the destination CouchDB. The number of documents created depends on the amount of the data presented in each file generated through the MySQL queries. If all of these conditions are met and run successfully then the desired data migrates to the CouchDB and the operation completes successfully.

### **7.2.1.2 Data migration from multiple tables in multiple CouchDB documents**

In this scenario the data will come from multiple tables. This means that many join operations are part of the SQL script written to export data from the MySQL DB. These joins are used to join all the tables necessary for the destination data in the CouchDB. The rest of the process is same as already described in section 7.2.1.1 above. Also others details have already been covered in Chapter 6.

Goal G2 of this thesis was “**G2:** Second goal is to ensure that identical operations can be performed against both databases and get identical results”. This goal relates to the question if the migrated data in CouchDB can be used for identical operations performed in the MySQL DB and can produce identical results. Below is how we will evaluate this goal.

### **7.2.2 Can the same operation be performed in both systems and achieve the same result?**

There are various operations performed in databases. Main four operations are select, insert, update, and delete. These operations will be performed on the identical data after the DM is done as specified in in step 7.2.1.1 and 7.2.1.2 above. These operations will be performed using the GUI application designed for the conference system while connected with the MySQL database and then the same operations will be performed again with the CouchDB using the GUI designed for the conference system while connected with the CouchDB after data migration.

A GUI tool has been designed for the conference system to communicate with both the MySQL and the CouchDB. The GUI tool connects to both MySQL and CouchDB and performs the same operations on the same data. The DMT is a part of this GUI tool. The results of the performed actions will confirm whether these operations were performed successfully and whether they produced the same results on the migrated data.



The third goal was “**G3:** third goal is to compare the performance of various operations performed in identical conditions in two systems”. The evaluation of this goal will be done after the first two goals are achieved successfully. This is done as follows:

### **7.2.3 What is the speed difference for same operation performed on both systems?**

The data migration is the main aspect of this thesis but one of the goals is to compare performance while performing the same operations on each system. A time estimate for performing SELECT, INSERT, UPDATE, and DELETE operations is a helpful tool to give the idea about the efficiency of both systems about each operation. The time will be noted for performing operations in the MySQL DB using the GUI tool and then same operations will be performed against CouchDB and time will be noted again. The difference in time will be helpful to establish the performance of two databases.

The fourth and final goal of this thesis is “**G4:** The fourth goal of this research is to determine the complexity of the code written for performing same operations in RDBMS and NoSQL.”. The evaluation of this goal is planned as follows:

### **7.2.4 How complex is the code for the same operation in both systems?**

Scripts are written for both systems to perform operations in both systems as discussed earlier. The code complexity will be measured by comparing the PHP scripts written for both systems for the same operations. The number of lines is the baseline for measuring the code complexity and the comparison will give a general idea how much more difficult it is to write code for the MySQL DB versus the CouchDB.

## CHAPTER 8

### RESULTS

The experiments and results have been conducted in keeping with the goals set for the thesis. Experiments have been conducted for migrating data from single and multiple tables, performing identical operations to ensure that the results are same, recording the speed of various operations performed in both the MySQL DB and CouchDB, and comparing the scripts written for complexity of the code in terms of number of lines per script for the identical operation. The next section contains the details of these experiments.

#### **8.1 Has the data migrated successfully?**

To answer this question actual data migration was performed from the relational MySQL DB into the NoSQL CouchDB. The environment was set for the data migration as specified in the implementation section. The migration was done in few steps as follows:

##### **8.1.1 Data population and extraction in MySQL**

Several scripts were written as the first step to create the database structure according to the requirements of the conference system. These scripts defined the structure and data model of the CS system. These scripts were then run to create database objects in the MySQL DB. Various conference system GUI forms were used to populate data in the database tables. The results of the data population were verified manually by going through each chunk of data.

Once the data population was done and data integrity was confirmed, the next step was to determine about the document structure in CouchDB. Determining CouchDB document structure and requirements enabled the writing of suitable queries that mapped the data in the MySQL DB to the document structure in CouchDB. Several scripts were written to export data into CSV files

according to the various document structures required in the CouchDB. These scripts were then run to export data successfully. The next step was to migrate the exported data into the CouchDB documents.

### **8.1.2 Data migration into the CouchDB**

Two possibilities were discussed earlier for the data migration. One was that all the data was migrated from a single MySQL table into CouchDB documents and the other possibility was that the data was migrated from multiple MySQL tables into CouchDB documents. These two possibilities were tested as follows.

#### **8.1.2.1 Data migration from single MySQL table into CouchDB documents**

The CSV files generated above were optionally modified to include the meaningful unique document id instead of using the system generated meaningless ids as mentioned earlier. When the CSV files were ready for data import, the data migration tool which is a part of the conference system application was used to create the documents and import data into the CouchDB documents.

Individual scripts were used to import data for each different type of document. The document structure for each document was defined in the PHP script code blocks which resulted in several scripts matching the required document structures in CouchDB. The order of the document fields was mapped to the fields of exported CSV files so that correct data import occurs in respective fields.

The DMT was used to select appropriate scripts to create documents and import data in the CouchDB database. Figure 8-1 below presents the part of utility used for data migration.

Please use the browse button below to choose an import file:

Figure 8 - 1: Data migration tool

To start the data migration from the selected file, the DMT allows choosing the correct file for data migration using ‘Browse...’ button and then migrate the data into the CouchDB using the ‘Import’ button allows starting the data migration from the selected file.

To confirm that the data migrated successfully consider the information in the conference system USERS’ table. The USERS table contains information of all users in the CS system. The information was retrieved in the CS system GUI tool. This information is shown in the figure 8-2 below from a form into CS system GUI tool.

First Name	Last Name	User Name	Email	Role	Edit	Delete
Instructor	Instructor	instructor	instructor@usask.ca	instructor	<a href="#">Edit</a>	<a href="#">Delete</a>
Marker	Marker	marker	marker@usask.ca	marker	<a href="#">Edit</a>	<a href="#">Delete</a>
Muhammad	Mughees	muhammad	muh@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
John	Doe	johdoe	john@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
Ala	Bill	alabill	ala@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
Mark	Cam	markcam	mark@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>

Figure 8 - 2: CS form showing data from single MySQL table

To migrate the 'USERS' table into the CouchDB documents a script was written and run in the MySQL database as defined in the Chapter 7 section 7.2.1.1. The script run generated a file with data ready to be migrated into the CouchDB.

Then the data migration tool shown in Figure 8-1 was used to select the file to migrate the data. The migrated data in the CouchDB for Users is shown below in Figure 8-3.

<b>First Name</b>	<b>Last Name</b>	<b>User Name</b>	<b>Email</b>	<b>Role Name</b>
ala	bill	alabill	ala@usask.ca	student
instructor	instructor	instructor	instructor@usask.ca	instructor
john	doe	johndoe	john@usask.ca	student
mark	cam	markcam	cam@usask.ca	student
maker	marker	marker	marker@usask.ca	marker
muhammad	mughees	muhammadmughees	muh@usask.ca	student

Figure 8 - 3: Migrated data from USERS table inside CouchDB documents

It is obvious from the comparison between figures 8-2 and 8-3 that the data migrated successfully and identical results were retrieved which proves the success of the data migration tool.

### 8.1.2.2 Data migration from multiple MySQL tables into the CouchDB documents

The second scenario mentioned earlier was the data migration from multiple MySQL tables. To test this scenario experiments were made by writing various scripts in the MySQL DB that created the CSV files containing data from the MySQL tables. These files were then used for migrating data into the CouchDB documents as already explained in the Chapter 7 section 7.2.1.1.

In the conference system built for this thesis, students submit their assignments and then the instructor/marker assign those documents to another student for review. Figure 8-4 below is another form from the CS application that shows the ‘Reviewer’ and the document assigned to a reviewer. This information can also be updated or deleted. This information is coming from three different tables in the conference system namely USERS, DOCUMENTS, and ASSIGN\_DOC. Different joins are used to get the required information from the tables.

Reviewer	Document Name	Change Reviewer	Delete
Muhammad	alabill_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
Muhammad	johndoe_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
Muhammad	markcam_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
johndoe	alabill_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
johndoe	markcam_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
alabill	johndoe_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
alabill	markcam_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
markcam	alabill_assign3	<a href="#">Update</a>	<a href="#">Delete</a>
markcam	johndoe_assign3	<a href="#">Update</a>	<a href="#">Delete</a>

Figure 8 - 4: Documents assigned to reviewers in MySQL

The same data was migrated to the CouchDB conference system as explained earlier for data migration from multiple tables. Figure 8-5 below shows a screenshot from the CS system application form with the data populated for the same documents and reviewers' information as considered in Figure 8-4 above.

<b>Reviewer</b>	<b>Document Name</b>	<b>Change Reviewer</b>	<b>Delete</b>
johndoe	alabill_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
Muhammad	alabill_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
markcam	alabill_assign3	<a href="#">Update</a>	<a href="#">Delete</a>
alabill	johndoe_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
Muhammad	johndoe_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
markcam	johndoe_assign3	<a href="#">Update</a>	<a href="#">Delete</a>
Muhammad	markcam_assign1	<a href="#">Update</a>	<a href="#">Delete</a>
alabill	markcam_assign2	<a href="#">Update</a>	<a href="#">Delete</a>
johndoe	markcam_assign2	<a href="#">Update</a>	<a href="#">Delete</a>

Figure 8 - 5: Documents assigned to reviewers in CouchDB

A comparison between the data from Figures 8-4 and Figures 8-5 it shows that the identical data was retrieved for the identical operation in both the MySQL and the CouchDB databases. This also proves the authenticity of the data and the data migration success.

After the data was migrated successfully from the relational MySQL to NoSQL CouchDB, the verification of migrated data was performed manually. Manual confirmation of data was enough to a certain extent. To ensure that the data was migrated successfully the next section demonstrates that several operations including SELECT, INSERT, UPDATE, and DELETE were performed on the migrated data successfully.

## **8.2 Can the same operation be performed in both systems and achieve the same results?**

To get the answer for this question several identical experiments were performed in both MySQL and CouchDB systems using the CS application. First an operation was performed when connected to the MySQL with the CS application and then same operation was performed while connected to CouchDB. The results were then compared to verify that the same results were achieved in both systems. The next section discusses various scenarios used for the experiments and their results.

### **8.2.1 Verify data after SELECT operation**

To verify if SELECT operations could be performed successfully producing identical results, various forms were used to see the data. The figures below shows that the results achieved in both MySQL and NoSQL systems were identical which proves the data migration was successful and the migrated data is correct.



Earlier in Figure 8-5 the information about the CS documents and respective reviewers was presented. Figure 8-6 below presents the information about the author and the documents this time. The application form in this figure shows the author and their documents and option to assign documents for review. This information is coming from multiple MySQL tables.

Author	Document Name	Assign
alabill	alabill_assign1	<a href="#">Assign</a>
alabill	alabill_assign2	<a href="#">Assign</a>
alabill	alabill_assign3	<a href="#">Assign</a>
johndoe	johndoe_assign1	<a href="#">Assign</a>
johndoe	johndoe_assign2	<a href="#">Assign</a>
johndoe	johndoe_assign3	<a href="#">Assign</a>
markcam	markcam_assign1	<a href="#">Assign</a>
markcam	markcam_assign2	<a href="#">Assign</a>
markcam	markcam_assign3	<a href="#">Assign</a>

Figure 8 - 6: Documents and author information in MySQL

The SELECT operation in the same application form of the migrated data in the CouchDB is presented in Figure 8-7 below.

<b>Author</b>	<b>Document Name</b>	<b>Assign</b>
alabill	alabill_assign1	<a href="#">Assign</a>
alabill	alabill_assign2	<a href="#">Assign</a>
alabill	alabill_assign3	<a href="#">Assign</a>
johndoe	johndoe_assign1	<a href="#">Assign</a>
johndoe	johndoe_assign2	<a href="#">Assign</a>
johndoe	johndoe_assign3	<a href="#">Assign</a>
markcam	markcam_assign1	<a href="#">Assign</a>
markcam	markcam_assign2	<a href="#">Assign</a>
markcam	markcam_assign3	<a href="#">Assign</a>

Figure 8 - 7: Documents and authors information in CouchDB

Comparison of the figure 8-6 with the figure 8-7 clearly shows identical data for the SELECT operation performed in the same CS system application form.

### 8.2.2 Verify data after the UPDATE operation

The figures 8-8 and 8-9 below shows records before and after an UPDATE operation in the CS system application form for CouchDB.

The figure 8-8 below shows the users' information in CouchDB.

First Name	Last Name	User Name	Email	Role Name	Edit	Delete
ala	bill	alabill	ala@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
instructor	instructor	instructor	instructor@usask.ca	instructor	<a href="#">Edit</a>	<a href="#">Delete</a>
john	doe	johndoe	john@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
mark	cam	markcam	cam@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
maker	marker	marker	marker@usask.ca	marker	<a href="#">Edit</a>	<a href="#">Delete</a>
muhammad	mughees	muhammadmughees	muh@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>

Figure 8 - 8: CouchDB Users' information

The next figure 8-9 shows the same form after the 'email' field for user 'Muhammad' was changed to 'muhammad@usask.ca'.

First Name	Last Name	User Name	Email	Role Name	Edit	Delete
ala	bill	alabill	ala@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
instructor	instructor	instructor	instructor@usask.ca	instructor	<a href="#">Edit</a>	<a href="#">Delete</a>
john	doe	johndoe	john@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
mark	cam	markcam	cam@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
maker	marker	marker	marker@usask.ca	marker	<a href="#">Edit</a>	<a href="#">Delete</a>
muhammad	mughees	muhammadmughees	muhammad@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>

Figure 8 - 9: CouchDB record after update operation

It is obvious that the record was updated successfully which shows that the UPDATE operation can be performed without any problem on the migrated data in the CouchDB.

### 8.2.3 Verify data after DELETE operation

The record for user ‘Muhammad’ was deleted from the application form shown in the figure 8-9 above to test the DELETE operation on the migrated data. The figure 8-10 below shows after the record was deleted.

First Name	Last Name	User Name	Email	Role Name	Edit	Delete
ala	bill	alabill	ala@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
instructor	instructor	instructor	instructor@usask.ca	instructor	<a href="#">Edit</a>	<a href="#">Delete</a>
john	doe	johndoe	john@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
mark	cam	markcam	cam@usask.ca	student	<a href="#">Edit</a>	<a href="#">Delete</a>
maker	marker	marker	marker@usask.ca	marker	<a href="#">Edit</a>	<a href="#">Delete</a>

Figure 8 - 10: Users' information in CouchDB after the DELETE operation

The figure 8-10 shows that the record for user 'Muhammad' disappeared after the DELETE operation which proves that DELETE operation can be performed successfully on the migrated data in the CouchDB.

The test for the INSERT operation is in fact not required because all the data migration was actually an INSERT operation.

One of the goals of data migration was to migrate the data successfully without any data corruption to ensure data integrity. All the tests discussed above produced the expected results without loss of any data which proves that the DMT tool is working as expected and producing the correct results.

### 8.3 Compare the speed of various operations

The third goal of this thesis was to compare the speed of various operations such as SELECT, INSERT, UPDATE, and DELETE on the migrated data from the MySQL DB to the Couchdb.

These operations were run against the same data. For clarity and simplicity the operations were run against a single table of records. Experiments were carried out with five different sets of rows in the MySQL table and equal number of documents in the CouchDB. All four operations were performed in both the MySQL DB and the CouchDB while connected using the conference system application. The results of these operations are presented in the next section.

**8.3.1 SELECT operation time comparison**

The select operation was run against a different set of rows. The results are produced in the following Table 8-1.

<b>No of records</b>	<b>MySQL (time in seconds)</b>	<b>CouchDB (time in seconds)</b>
10	0	13
20	0.01	24
30	0.01	34
40	0.02	42
50	0.02	47

Table 8 - 1: SELECT operation times comparison in tabular form

Figure 8-1 below illustrates these records in graphical form. The time required to run the SELECT operation for different sets of data was almost close to zero in the MySQL whereas the CouchDB consumed much higher times to produce the same results for the same amount of data.

The results show that, the time for data retrieval kept increasing in both the CouchDB and the MySQL database. However, the time increase of the CouchDB is much higher than the time

increase in MySQL. Results also show that the increase in data retrieval time with the increasing number of rows in the MySQL database is quite steady and linear. However, this is not the case with the CouchDB. As the number of documents increases in the CouchDB, it takes more time to retrieve data but it is not a linear relation. The increase in time is not directly proportional to the increase in the data in the form of the number of documents. The increase in retrieval time for each additional ten documents i.e. 20, 30, 40, and 50 is 84%, 42%, 24%, and 12% respectively which actually shows that the rate of increase in time is decreasing with the increase in data. This is a good indication that the performance of CouchDB gets better with increasing amount of data. This also implies why CouchDB may perform better with huge amount of data.

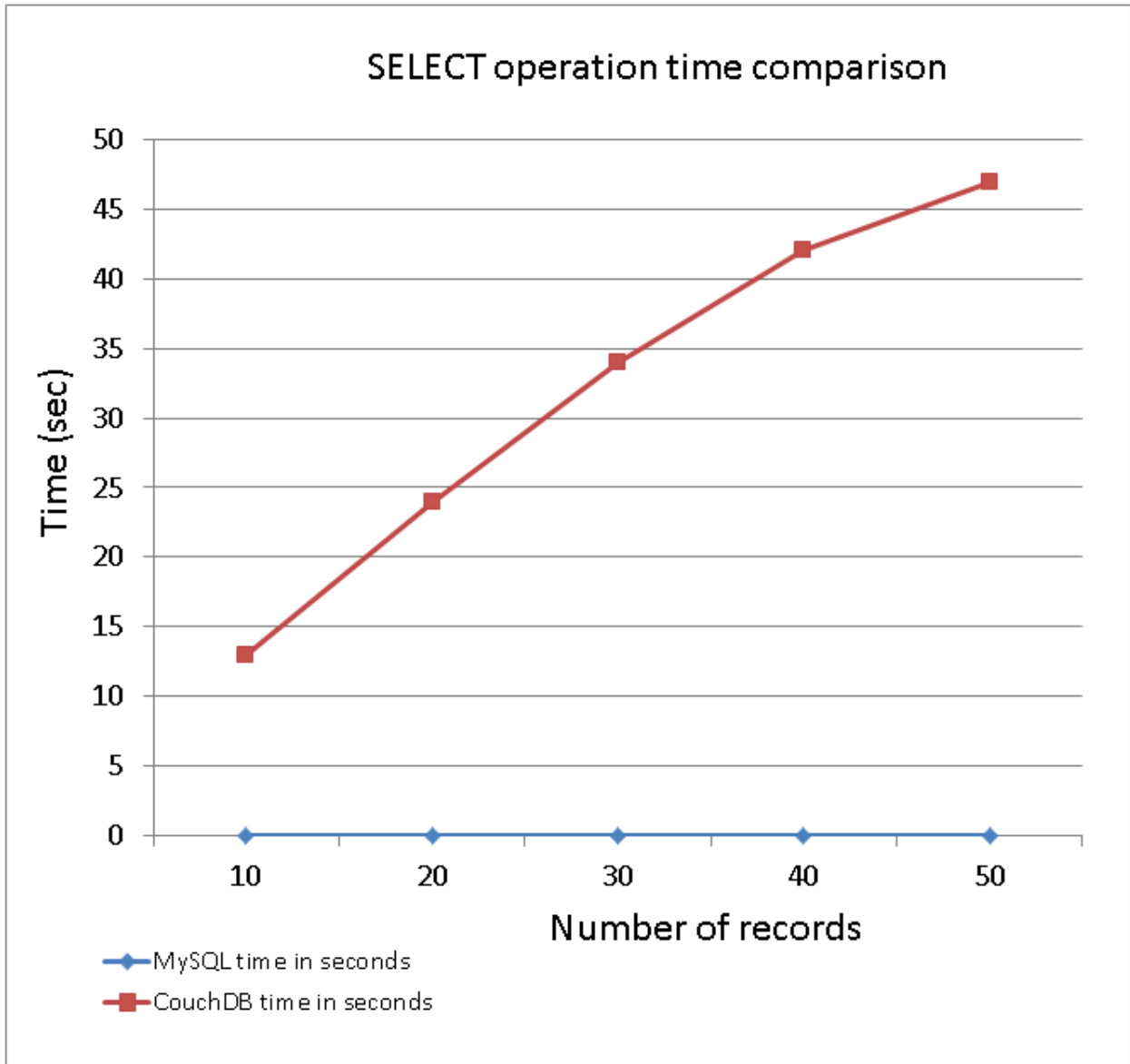


Figure 8 - 11: SELECT operation time comparison

### 8.3.2 INSERT operation time comparison

The INSERT operation was also run against different sets of rows. The same sets of rows were used for both the MySQL DB and the CouchDB. The results are produced in the following Table 8-2.



<b>No of records</b>	<b>MySQL (time in seconds)</b>	<b>CouchDB (time in seconds)</b>
10	1	9
20	1.1	21
30	1.3	32
40	1.43	43
50	1.78	54

Table 8 - 2: INSERT operation times comparison in tabular form

Figure 8-12 below shows that the time required to run the INSERT operation for different sets of data was much less against the MySQL DB than the time taken by the CouchDB against the same number of records. As was the case with the SELECT operation, time taken by CouchDB kept increasing with the increasing number of documents but unlike the SELECT statement the time was directly proportional to the increase in number of documents. This shows a steady linear time increase with the increasing number of documents to insert. This trend indicates that unlike the SELECT statement, the proportionate INSERT time in the CouchDB does not decrease with increasing number of documents.

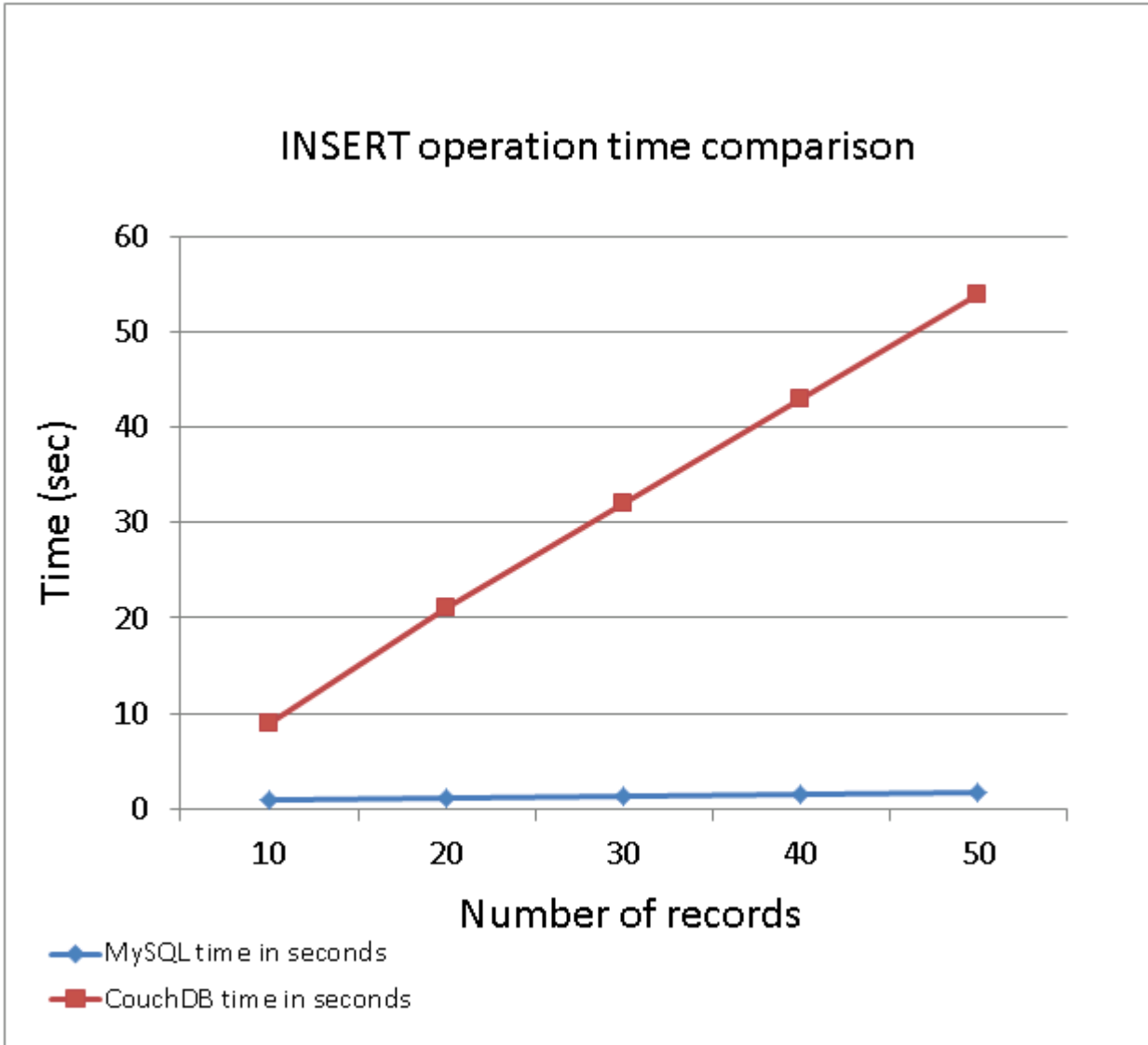


Figure 8 - 12: INSERT operation time comparison

### 8.3.3 UPDATE operation time comparison

The same number of sets and the same number of rows in each set were used in the UPDATE experiments as were used in the SELECT and INSERT operations. The results obtained are presented below in Table 8-4.

<b>No of records</b>	<b>MySQL (time in seconds)</b>	<b>CouchDB (time in seconds)</b>
10	0.001	60
20	0.001	110
30	0.005	162
40	0.007	208
50	0.01	246

Table 8 - 3: UPDATE operation times comparison in tabular form

Figure 8-3 above shows that the UPDATE operation also followed the same trend already seen with SELECT operation. The time taken by the MySQL DB for update was much shorter than the time taken by the CouchDB to update same amount of data. Again, the update time for the MySQL DB was almost close to zero while CouchDB consumed much higher time to update the same amount of data.

Like the SELECT operation, results show that the increase in time for update with the increasing number of rows in the MySQL database is quite steady and linear. However, this is not the case with the CouchDB. As the number of documents increases in the CouchDB, it takes more time to update data but it is not linear relation. The increase in time is not directly proportional to the increase in the data. The increase in update time for each additional set of ten documents i.e. 20, 30, 40, and 50 documents is 63%, 47%, 28%, and 18% respectively which actually shows that the rate of increase in time is decreasing gradually with the increase in data.

This also indicates that the update performance of the CouchDB may actually improve with the increase in data.

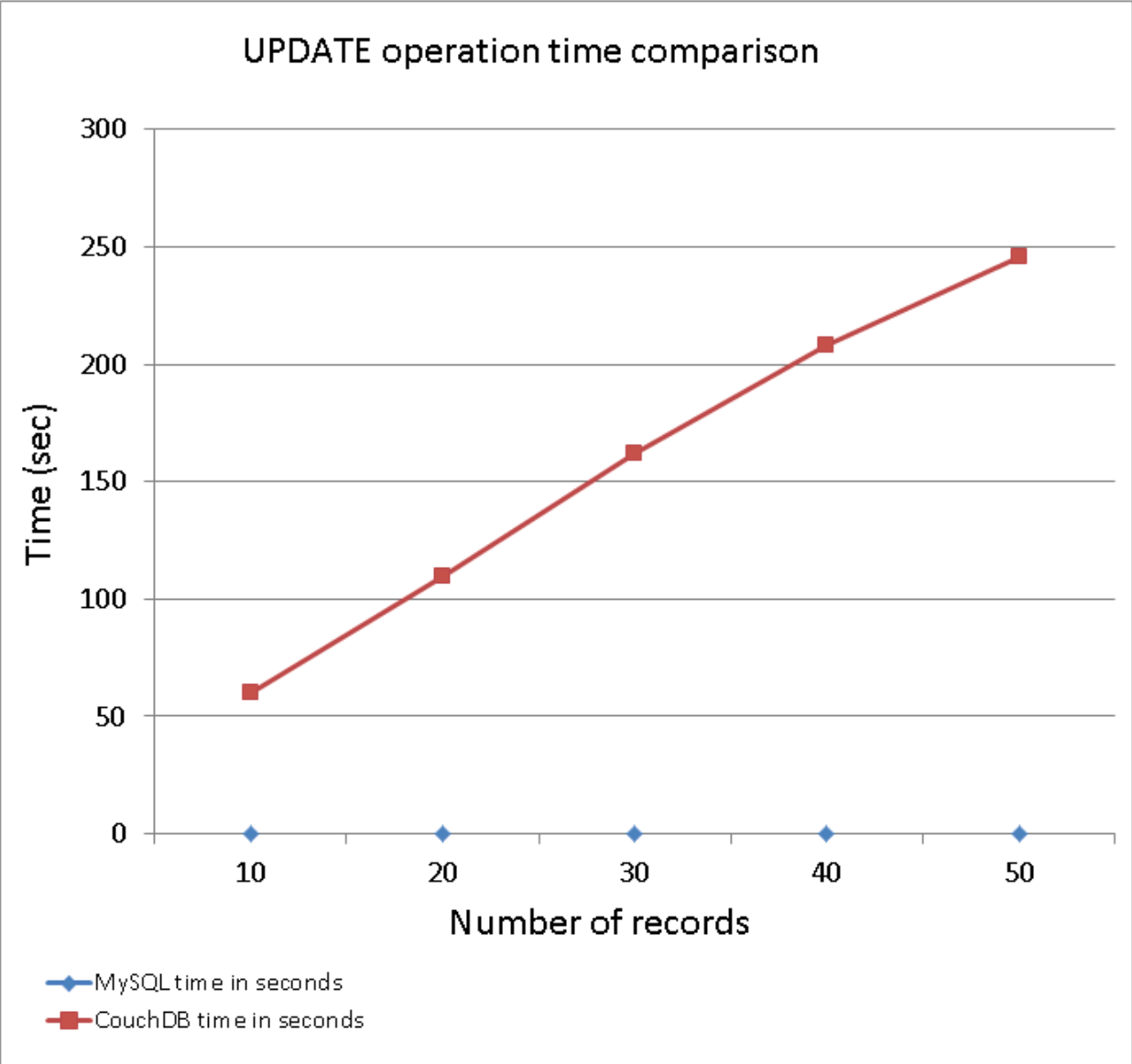


Figure 8 - 13: UPDATE operation time comparison

### 8.3.4 DELETE operation time comparison

The DELETE experiment was also tested against the same number of sets and each set consisted of same number of records in case of the MySQL and same number of documents in the case of the CouchDB. The results of the DELETE operation are very much the same as observed in the case of the other three operations already discussed. The results of the DELETE operation are presented in table 8-4 below.

<b>No of records</b>	<b>MySQL (time in seconds)</b>	<b>CouchDB (time in seconds)</b>
10	0.031	50
20	0.047	110
30	0.075	160
40	0.077	207
50	0.078	247

Table 8 - 4: DELETE operation times comparison in tabular form

Finally the DELETE operation behaved exactly in the same manner as was the case with SELECT and UPDATE. Figure 8-3 below shows that the DELETE operation also followed the same trend already seen with SELECT and UPDATE operations. The time taken by the MySQL DB to delete data was much shorter than the time taken by the CouchDB to delete the same amount of data. Again, the delete time for the MySQL DB was almost close to zero while CouchDB consumed much higher time to delete the same amount of data.

Like the SELECT and UPDATE operations, results show that the increase in time to delete data with the increasing number of rows in the MySQL database is quite steady and linear. However, this is not the case with the CouchDB. As the number of documents increases in the CouchDB, it takes more time to delete data but it is not a linear relation. The increase in time is not directly proportional to the increase in the data. The increase in delete time for each additional set of ten documents i.e. 20, 30, 40, and 50 documents is 120%, 45%, 29%, and 19% respectively which actually shows that the rate of increase in time is decreasing gradually with the increase in data. This also indicates that the delete performance of the CouchDB may actually improve with the increase in data.

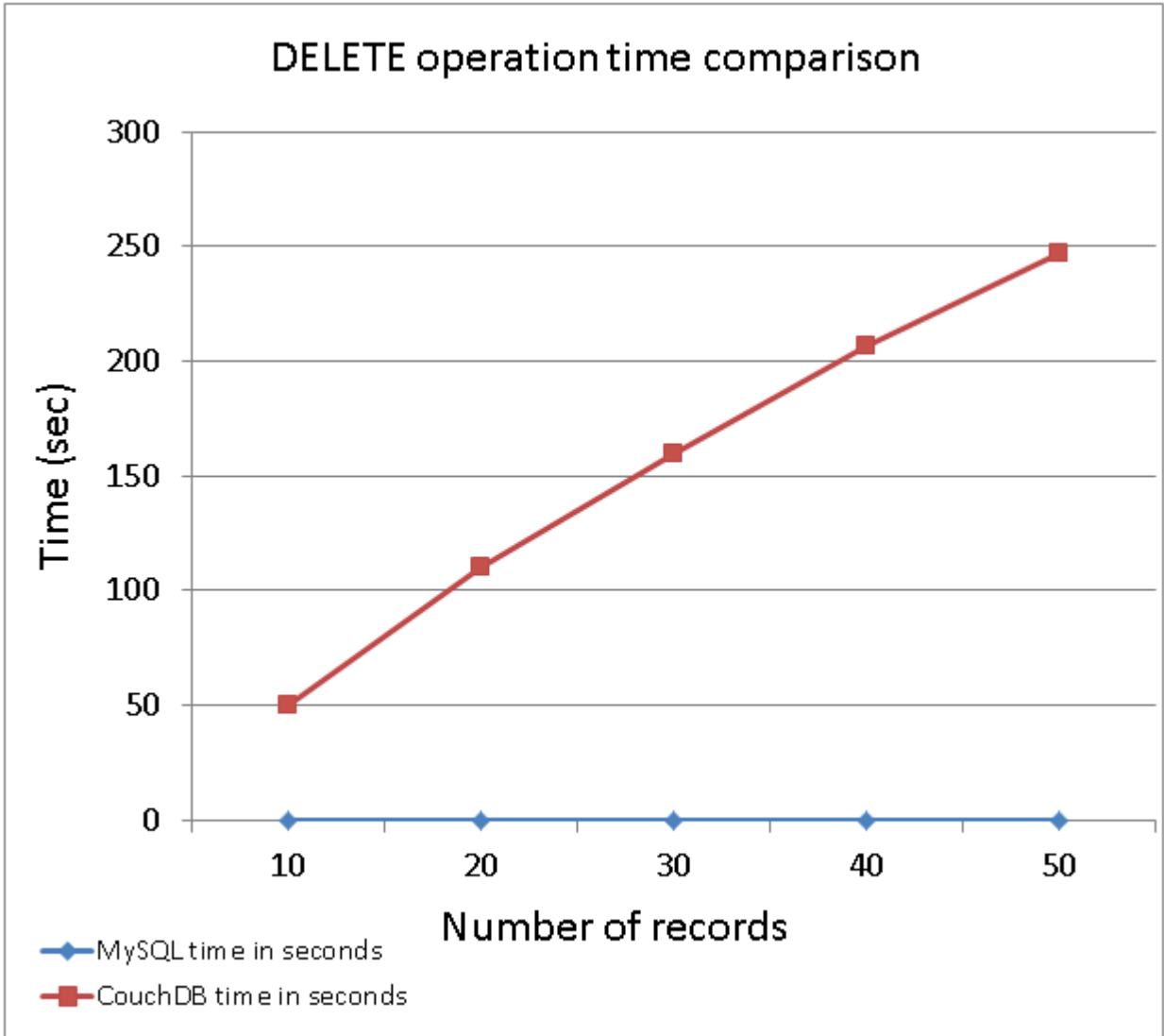


Figure 8 - 14: DELETE operation time comparison

#### 8.4 Determine the code complexity

As mentioned before the code complexity is determined by the number of lines of code for performing the same operation in both the systems i.e. the MySQL DB and the CouchDB.

The same rules were applied to scripts for both the MySQL DB and the CouchDB systems. Only the code for performing the same operations was compared. Identical code in some sections of scripts written for each system has not been counted in the table below.

Table 8-5 below shows a comparative study about the code complexity of scripts written and implemented for CS system in the MySQL DB and the CouchDB.

Script	Number of lines in MySQL	Number of lines in CouchDB
add_doc	13	60
add_reviewed_doc	16	58
all_docs	133	179
assign_docs	19	24
author_reply_comments	12	34
check_role	5	8
check_valid_user	5	8
comments	8	14
confirmDelete	17	50
confirmDeleteAssignedDoc	17	34
confirmDeleteAssignment	16	53
create_users	17	20
download_file	35	69



download_rev_file	5	17
edit_marks	22	49
edit_users	30	36
import_users	19	17
manage_docs	124	165
manage_marks	44	55
map_assignee_doc	41	62
post_assignment	7	11
survey	37	70
update_assignments	43	61
update_map_assignee_doc	42	88
update_users	99	78
upload	24	27
upload_reviewed_doc	42	34
view_assigned_docs	19	24
view_assignment	15	31
view_evaluations	21	28
view_reviews	28	42

Table 8 - 5: Comparative study of code complexity

The graph in the figure 8-15 below gives a graphical view of the information.

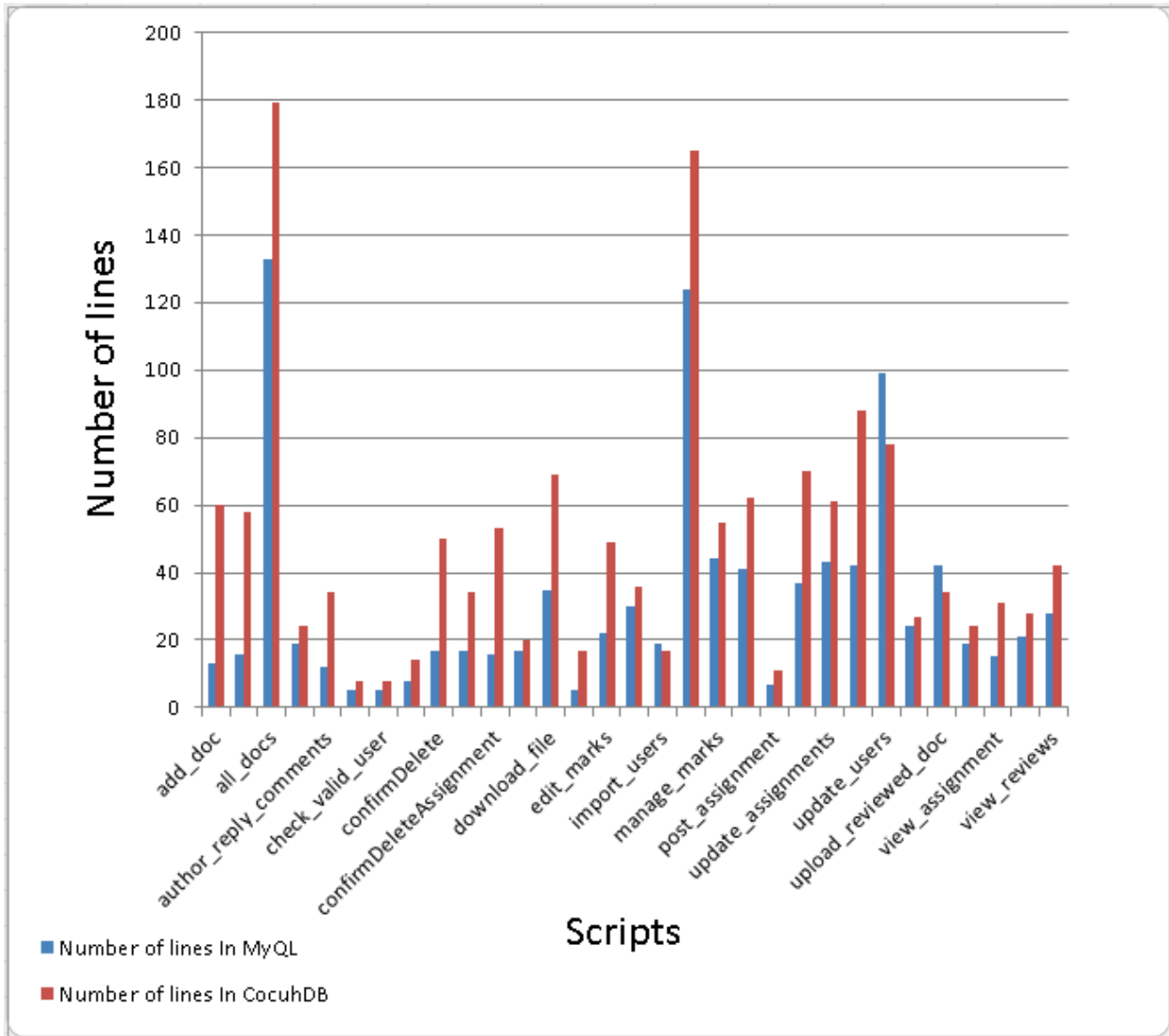


Figure 8 - 15: Code complexity comparison

The graph clearly shows that number of lines of code required for performing the same operation is generally more in CouchDB than the counterpart MySQL DB. This could mean that more resources and maintenance would be required to generate, run, and maintain code written for the CouchDB than the MySQL DB.

Generally NoSQL databases are considered better than the RDBMS databases but the results in this thesis negate this impression. The reality is that the NoSQL databases have been designed for terabytes of data, millions of users and the distributed environment in a cloud.

The key attributes of NoSQL databases are scalability and flexibility. These are the two main characteristics of NoSQL databases that have made them the preferred choice for big data projects in CC environments. NoSQL databases offers highly scalable environment because they have been designed to store and manage data in a distributed environment consisting of clusters of cheap commodity servers. The distributed environment set up in this way offers high scalability by adding more machines seamlessly.

The advantage of flexibility is that it allows the modification of data structure during system usage whenever required due to schema less nature of data. The RDBMS data model is rigid and becomes more difficult to evolve over time with the growth in data. This is not the case with The NoSQL database, however. The schema free and unstructured nature of the data model makes it easy to incorporate changes without disturbing the rest of the database objects e.g. documents. These NoSQL database characteristics contribute towards better performance at scale.

However, the NoSQL database implementation for this thesis did not provide the ideal environment suited to NoSQL databases. The lack of a distributed environment prevented the

NoSQL CouchDB from taking full advantage of the features they offer and thus resulted in a bad performance.

## CHAPTER 9

### CONCLUSION AND FUTURE WORK

#### 9.1 Conclusion

The modern approach in both grid and cloud computing has increased the demand for the NoSQL model of database. The need for seamless and rapid scalability, high availability and efficient databases are all contributing factors in this transition. However, storage requirements of the NoSQL databases are entirely different from that of the RDBMS databases.

The data storage in the RDBMS DBs using relations and normalization has been explained while different types of more popular NoSQL databases (key-value, document, columnar, and graph) has been presented.

The process of data migration becomes more challenging and interesting due to the entirely different data models of the RDBMS and the NoSQL databases. The real life data migration projects encounter simple to very complex scenarios. An example of a simple scenario is migrating data from one database into another database of the same vendor e.g. an earlier version of Oracle to a later version of an Oracle database or migrating in the same version but on a different platform e.g. from an Oracle database on Windows server to an Oracle database on UNIX server. Another example of slightly more complex data migration is from one RDBMS DB to another RDBMS DB e.g. migrating data from Oracle to SQL Server or vice versa. Very complex systems consist of much different types of data storage resources and data that is shared among them through several applications. The data in these systems is extracted on a nightly basis from several databases using Extract, Transform, and Load (ETL) tools and loaded into a data warehouse used for decision support system. These are usually heterogeneous systems consisting of different flavors of databases such as Oracle, SQL Server, DB2 etc.

Several approaches can be used for migrating data from source to target systems depending on the setup and requirements of the source and destination systems. These approaches include but are not limited to the use of an existing data migration tool, ETL technology, hand coding, and replication.

If a data migration tool meets all the current requirements of data migration then it is preferred to use this tool for a fast and accurate process. There are several high quality tools available for data migration between RDBMS systems that are used frequently as part of data migration strategy. There are also tools available for data migration across different platform e.g. from the database on Windows to the database on UNIX system.

For very complex heterogeneous systems consisting of different flavors such as Oracle and SQL Server on separate platforms, data migrates from one vendor to another vendor and to another platform. For these types of heterogeneous systems ETL (extract, transform, and load) tools are very helpful for data migration and used heavily. However, these tools also require manual changes termed ‘hand coding’ to meet the data migration requirements. Hand coding may be required for various reasons e.g. to gain more control over code to change current behavior or add more functionality.

Replication is another method of data migration that is usable in specific circumstances e.g. when both source and destination systems need to be available during data migration. Replication is usually helpful in situations when data conversion is not required and data is flowing between same vendors of database.

None of the solutions presented above meets all our requirements per say for data migration requirements from the RDBMS to the NoSQL databases. The approach followed in this thesis specifies an architecture that suits the demands of migrating data from the RDBMS to the

NoSQL databases. Due to simple nature of the CS system data model, data cleansing is not required because no data type conversion is required during data migration. However, this step may have more value with a more complex data model and can be accommodated in future work.

This research presents data migration from a MySQL to a document based CouchDB which is an extension of key-value DB type. This allows an easy mapping of data structure from the RDBMS to the NoSQL databases. This also allows storing data in a semi-structured format.

RDBMS systems store data in several tables in the RDBMS systems and use join operations for extracting required information. The data migration approach takes advantage of data extraction SQL code executed in the RDBMS and converts the data into the CouchDB documents by mapping the extracted fields to the CouchDB document. The approach has been proved with experimental results included in Chapter 8.

The results show that the data migration was achieved successfully. However, the secondary results of goal 2 and goal 3 suggests that the MySQL DB is better than CouchDB. The results also suggest that the code writing is more complex for CouchDB as compared to MySQL in terms of number of lines. This may not be reflecting the true picture as the CouchDB is more suitable in a distributed environment with replication features enabled. The power of CouchDB relies on its usage in a distributed environment and replicating data across participating nodes. A justified and true performance comparison with both systems deployed in distributed environment and taking advantage of the replication feature may provide promising results in favor of CouchDB.

The comparison of code complexity is more subjective in the sense that a lot depends on the personal coding skills of the programmer. A programmer with better SQL programming skills will write a more efficient code for the MySQL database as compared to writing the code for performing the same operation in the CouchDB. Similarly a programmer with better programming skills in the NoSQL environment and with better understanding of the programming language will write more efficient code for the CouchDB as compared to writing the code for performing same operation in the MySQL DB.

Another important factor in the code complexity is the selection of the programming language. Code written in JAVA may prove more efficient than using the PHP language. Use of different languages in different scenarios may provide a different level of performance and complexity.

Integrity of migrated data and the ability to use it for the same purpose as it was being used in the source RDBMS database is another important aspect of the data migration. For this purpose two version of same GUI tools were designed to work with MySQL and CouchDB respectively. Using these GUI tools the same operations were performed against the source MySQL database and the migrated CouchDB. The results included in Chapter 8 confirm that same results were achieved with these operations.

The NoSQL databases are usually considered faster than the RDBMS databases. A time based performance test was carried out against both systems running the same operations against the same amount of data. The results did not confirm the general impression about the speed of the NoSQL database as compared with relational database. The experiments were carried out in identical environments with same resources. However, data modeling and application coding



also plays an important role in measuring the performance and a different approach towards data modeling and application coding has the tendency to impact database performance.

The application maintenance also depends upon the code complexity. The maintenance becomes easier when the code is not complex and vice versa is also true. This research presents the difference in the code complexity by comparing the number of lines of code required to perform an operation in the MySQL DB with the number of lines of code required to perform the same operation as in the CouchDB. The results in Chapter 8 show that more code lines are required in the CouchDB application than the MySQL application to perform the same operation. However, a different application coding approach or use of a different programming language may show results in favor of CouchDB in future work.

In conclusion, the RDBMS systems have their limitations when the data grows out of the capacity of one machine. High availability and scalability is possible but complex and very costly in the RDBMS DBs. The NoSQL database on the other hand is an easy answer to these problems that could be the driving force to migrate data from the standard SQL to NoSQL databases. The DMT designed and presented in this thesis is a contribution towards the solution of this highly anticipated demand in the future.

## 9.2 Future Work

This research only presents the preliminary work carried out in the direction of migrating data from the RDBMS to the NoSQL systems using MySQL DB and CouchDB as the guinea pig. There is room for continuous improvement to make the data migration easy and seamless to the end user. Some of the other possible future enhancements are as follows:

- Extend the scope of the DMT to a distributed environment. The current implementation of the conference system consists of a standalone implementation of the MySQL and the CouchDB databases. As discussed in the Conclusion Section 9.1 above, the results of the experiments carried out earlier show that the MySQL database is more superior as compared to the CouchDB database. But these results are applicable in a standalone implementation of these databases. However, the CouchDB works best when it is implemented in a distributed environment and the replication feature is enabled. The CouchDB takes advantage of replication keeping all participated nodes up-to-date with the latest information. Availability of the same information on more than one node enables applications to get the desired information quickly from one of these nodes based on how they are configured resulting in an improved performance. Future work in this direction will be helpful to enhance the results produced in this thesis.
- Expand the scope to include a more complex data model with complex data types. The CouchDB documents have the ability to handle multiple types of data, including arrays and nested objects. However, due to simple nature of the CS system data model, full advantage of nested objects could not be taken in this

research. This may have a negative impact on the performance of the CouchDB which can be tested in a future work by taking advantage of the nested objects.

- Include data types that need conversion. The data types used in the CS system in the MySQL DB had the corresponding data types in the destination CouchDB and didn't need conversion. The real life projects are much more complex and make use of several advanced data types. A future work including data types that need conversion from the RDBMS to the NoSQL DB will be helpful to observe the behavior of data migration and DB performance.
- Evaluate the performance in distributed environment with replication option enabled. The RDBMS database naturally works well in a standalone environment and a better performance of the CS system application with MySQL DB as compared to the CouchDB is not surprising. The CouchDB on the other hand is supposed to work well in a distributed environment with the replication feature enabled. A better choice for true comparison may be to compare an environment where the MySQL database is implemented standalone while the CouchDB is implemented in a distributed environment with replication feature enabled. An even more realistic approach would be to implement both the MySQL and CouchDB databases in a distributed environment with replication option enabled and then compare the performance.
- Use other rich languages to add more functionality and improve the DMT tool. This thesis used PHP as the programming language for the GUI tool. The code complexity using PHP seems to be in favor of the MySQL DB. A more comprehensive comparison by using other languages such as JAVA may improve

the results in favor of the CouchDB and may also be a contributing factor towards the CouchDB performance enhancement.

- Enhancement to communicate with the CouchDB and MySQL DB simultaneously. Currently the migration is done in two steps. First connect to the MySQL database using the MySQL Workbench or other client tool and extract the data using scripts and then connect to the CouchDB using DMT introduced in this thesis to migrate the data. The DMT tool has the room for future improvement to perform these tasks by simultaneously connecting to the MySQL and CouchDB DBs.
- Automatic metadata and data capturing. On the fly capture of the data and structure from RDBMS and convert them to the required CouchDB documents data format. Currently the data is first captured in CSV files by connecting to the MySQL database and issuing SQL code. The CouchDB documents structure is then designed based on the extracted fields from the MySQL database. This CouchDB document structure is then embedded in individual PHP scripts in order to create the document according to defined structure and populate with CSV file data. All this process of capturing the RDBMS table structure and converting it into the CouchDB document structure can be automated in a future enhancement.
- Generalize the tool to work with any NoSQL database. At present the tool is good for use with document structure based database. The same concept used for the document based structure for the designing of this tool can be used to expand the scope of this tool and make it compatible with other NoSQL databases.

- Enhance the GUI tool to automatically run the desired script for data migration in the CouchDB. Currently the link to each different type of CouchDB document needs to be updated in the PHP script to create the CouchDB documents and populate them with data every time the document structure changes. This process has great potential to automatically run the correct script without making manual changes in the link each time.
- Extend the scope of the utility to other platforms. Currently the DMT works in Windows environment but the scope can be extended to other platforms such as UNIX or LINUX in a future enhancement thus increasing the robustness of this tool.

## LIST OF REFERENCES

- [1] 1NF to 5NF-Normalization with Eg: 2011.  
*Available: <http://www.scribd.com/doc/49645421/1NF-to-5NF-Normalization-with-Eg>.*
- [2] 5 Graph Databases to Consider: 2011. *Available: <http://readwrite.com/2011/04/20/5-graph-databases-to-consider>.*
- [3] Apache Cassandra 0.7 Documentation: 2013.  
*Available: [http://www.datastax.com/docs/0.7/data\\_model/supercolumns](http://www.datastax.com/docs/0.7/data_model/supercolumns).*
- [4] Bermbach, D. and Tai, S. 2011. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. *Proceedings of the 6th Workshop on Middleware ...* (2011).
- [5] Borthakur, D., Rash, S., Schmidt, R., Aiyer, A., Gray, J., Sarma, J. Sen, Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D. and Menon, A. 2011. Apache hadoop goes realtime at Facebook. *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*. (2011), 1071.
- [6] Brewers CAP Theorem on distributed systems: 2010.  
*Available: <http://www.royans.net/arch/brewers-cap-theorem-on-distributed-systems/comment-page-1/>.*
- [7] Burckhardt, S., Gotsman, A. and Yang, H. 2013. Understanding Eventual Consistency. *dagstuhl.de*. (2013).
- [8] Calero, C., Ruiz, F. and Baroni, A. 2006. An ontological approach to describe the SQL: 2003 object-relational features. *Computer Standards & ...* (2006), 1–28.
- [9] Carlo, C., Jones, E.P.C., Papa, R.A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H. and Zeldovich, N. 2011. Relational Cloud: A Database-as-a-Service for the Cloud.
- [10] Cattell, R. 2011. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*. December (2011).
- [11] Chamberlin, D. and Boyce, R. 1974. SEQUEL: A structured English query language. *Proceedings of the 1974 ACM SIGFIDET ( ...)* (1974).
- [12] Chanchary, F.H. and Islam, S. 2012. Data Migration: Connecting Databases in the Cloud.
- [13] Chang, F., Dean, J. and Ghemawat, S. 2008. Bigtable: A distributed storage system for structured data. ... *on Computer Systems ( ...)* (2008).
- [14] Codd, E. 1971. Normalized data base structure: A brief tutorial. ... *SIGMOD) Workshop on Data Description, Access and ...* (1971).
- [15] Codd, E.F. 1970. A relational model of data for large shared data banks. 1970. *M.D. computing : computers in medical practice*. 15, 3 (1970), 162–166.

- [16] Codd, E.F. 1979. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*. 4, 4 (Dec. 1979), 397–434.
- [17] Codd, E.F. 1971. Further Normalization of the Data Base relational Model. *IBM Research Journal RJ909*. (1971).
- [18] Codd, E.F. 1974. Recent Investigations in Relational Database Systems. *IFIP Congress* (1974).
- [19] Compare NoSQL Databases: 2013. Available:<http://nosql.findthebest.com/>.
- [20] CouchDB the definitive guide: 2013. Available:<http://guide.couchdb.org/draft/consistency.html#figure/3>.
- [21] Date, C.J. 2000. *An Introduction to Database Systems - 7th ed.*
- [22] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. (2004).
- [23] Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P. and Vogels, W. 2007. Dynamo: amazon’s highly available key-value store. *SOSP*. (2007), 205–220.
- [24] Fagin, R. 1981. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*. 6, 3 (Sep. 1981), 387–415.
- [25] Fagin, R. 1977. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*. 2, 3 (1977), 262–278.
- [26] Fagin, R. 1979. Normal Forms and Relational Database Operators.
- [27] Getting started with Cassandra: 2010. Available:<http://nosql.mypopescu.com/post/573604395/tutorial-getting-started-with-cassandra>.
- [28] Getting started with CouchDB: meet PHP on Couch: 2010. Available:[http://devzone.zend.com/1782/getting-started-with-couchdb\\_meet-php-on-couch/](http://devzone.zend.com/1782/getting-started-with-couchdb_meet-php-on-couch/).
- [29] Gilbert, S. and Lynch, N. 2012. Perspectives on the CAP Theorem. *Computer*. 45, 2 (Feb. 2012), 30–36.
- [30] Graph Database: 2012. Available:<https://code.google.com/p/orient/wiki/GraphDatabase>.
- [31] Grier, D.A. 2012. The Relational Database and the Concept of the Information System. *IEEE Annals of the History of Computing*. 34, 4 (Oct. 2012), 9–17.
- [32] Groff, J.R., Weinberg, P.N. and Osborne, M. 2002. *SQL : The Complete Reference , Second Edition*.

- [33] Hetherington, T.H., Rogers, T.G., Hsu, L., O'Connor, M. and Aamodt, T.M. 2012. Characterizing and evaluating a key-value store application on heterogeneous CPU-GPU systems. *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. (Apr. 2012), 88–98.
- [34] Hodak William and Kumar, S. 2007. Oracle Database 11g Performance and Scalability Oracle Database 11g Performance and Scalability.
- [35] Introducing FlockDB: 2010. Available:<https://blog.twitter.com/2010/introducing-flockdb>.
- [36] Introduction to Kyoto Products: 2013. Available:<http://fallabs.com/kyotocabinet/kyotoproducts.pdf>.
- [37] ISO/IEC 9075-11:2011: 2011. Available:[http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=53685](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53685).
- [38] Kline, K.E. 2004. *SQL in a nutshell, 2nd Edition*, O'Reilly.
- [39] Konstantinou, I. and Angelou, E. 2011. On the Elasticity of NoSQL Databases over Cloud Management Platforms (extended version). (2011).
- [40] Lawson, P. and o' donoghue, M. 2007. Approaches to consent in canadian data protection law. (2007), 23–42.
- [41] LeDoux, C. and Jr, D.P. 1982. Reflections on Boyce-Codd normal form. ... *of the 8th International Conference on Very ...* (1982), 131–141.
- [42] Lennon, J. 2009. *Beginning CouchDB*.
- [43] Ling, T., Tompa, F.W. and Kameda, T. 1981. An improved third normal form for relational databases. *ACM Transactions on Database Systems*. 6, 2 (Jun. 1981), 329–346.
- [44] Making the Shift from Relational to NoSQL: 2013. Available:[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/Couchbase\\_Whitepaper\\_Transitioning\\_Relational\\_to\\_NoSQL.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/Couchbase_Whitepaper_Transitioning_Relational_to_NoSQL.pdf).
- [45] Makinouchi, A. 1977. A consideration on normal form of not-necessarily- normalized relation in the relational data model.
- [46] Map-Reduce and the New Software Stack: 2013. Available:<http://infolab.stanford.edu/~ullman/mmds/ch2.pdf>.
- [47] Migration of Relational Data structure to Cassandra (No SQL) Data structure: 2011. Available: <http://www.codeproject.com/Articles/279947/Migration-of-Relational-Data-structure-to-Cassandr>.
- [48] Navathe, S. 1992. Evolution of data modeling for databases. *Communications of the ACM*. 9 (1992).



- [49] NorthScale Enters Market, Addresses Exploding Cost of Web Application Data Management: 2013. Available:<http://www.prnewswire.com/news-releases/northscale-enters-market-addresses-exploding-cost-of-web-application-data-management-87768842.html>.
- [50] NoSQL and Document Oriented Databases: 2013. Available:<http://ruby.about.com/od/nosqldatabases/a/nosql1.htm>.
- [51] NoSQL: 2013. Available:<http://en.wikipedia.org/wiki/NoSQL>.
- [52] NoSQL: An Overview of NoSQL Databases: 2013. Available:<http://newtech.about.com/od/databasemanagement/a/Nosql.htm>.
- [53] Oracle berkeley db: 2010. Available:<http://www.oracle.com/technetwork/products/berkeleydb/berkeley-db-datasheet-132390.pdf>.
- [54] Relational Database Model Introduced by E.F. Codd (1970): 2013. Available:<http://people.scs.carleton.ca/~bertossi/DI12/material/Codd12Rules.txt>.
- [55] Rules of Engagement – NoSQL Column Data Stores: 2013. Available:<http://www.ingenioussql.com/2013/02/28/rules-of-engagement-nosql-column-data-stores/>.
- [56] Schram, A. and Anderson, K. 2012. MySQL to NoSQL: data modeling challenges in supporting scalability. *Proceedings of the 3rd annual conference on ....* (2012).
- [57] Seguin, K. 2011. *The Little MongoDB Book*. (2011).
- [58] Seguin, K. 2012. *The little Redis Books*. (2012).
- [59] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A. and Price, T.G. 1979. An Access Path Selection in a Relational Database Management System.
- [60] Sethuraman, P. 2009. Performance and Scalability of Microsoft ® SQL Server ® on VMware vSphere™ 4.
- [61] Simon, S. 2000. Brewer ’ s CAP Theorem. (2000), 1–6.
- [62] Symbol Table: 2013. Available:[http://en.wikipedia.org/wiki/Symbol\\_table](http://en.wikipedia.org/wiki/Symbol_table).
- [63] The CAP Theorem: Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services: 2010. Available: <http://www.slideshare.net/alekbr/cap-theorem>.
- [64] The time for NoSQL standards is now: 2012. Available:<http://www.infoworld.com/d/application-development/the-time-nosql-standards-now-205109>.
- [65] Ullman, J.D. 1989. *Principles of Database and Knowledge-Base Systems*.

- [66] Wei Hao, I.-L.Y. and B.T. 2009. Dynamic Service and Data Migration in the Clouds.
- [67] What is a Graph Database?: 2013.  
*Available:*<http://docs.neo4j.org/chunked/milestone/what-is-a-graphdb.html>.
- [68] White Paper Beating the CAP Theorem: 2011. *Available:*<http://www.geniedb.com/wp-content/uploads/2011/03/cap-theorem.pdf>.
- [69] Why NoSQL ? the database status quo: 2013.  
*Available:*<http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf>.
- [70] WTF IS A SUPERCOLUMN: 2013. *Available:*<http://arin.me/post/40054651676/wtf-is-a-supercolumn-cassandra-data-model>.

## APPENDIX A DATA MODEL OF CONFERENCE SYSTEM

### **A1. Users**

The “users” table stores information about all users in the system. This table has an auto increment primary key and a unique key for the ‘username’ attribute.

### **A2. Conferences**

This “conferences” table stores information about conferences and the owner of the conference.

### **A3. Documents**

The “documents” table stores information about all documents submitted by students. Document name, url, time created and other information goes to this table.

### **A4. Doc\_author**

The “doc\_author” table relates the ‘documents’ table with the ‘users’ table to find out the author for a specific document. The primary key in this table is composite in nature and composed of “doc\_id” and “user\_id” attributes.

### **A5. Reviewed\_documents**

The “reviewed\_documents” table stores information about document and the reviewer of the document. It also contains the information about ‘url’ of reviewed documents

### **A6. Review\_status**

The status of the reviewed documents or document under review goes to the “review\_status” table. Information about the document under review, reviewer, start and end time of review and whether review is still open or done is stored in this table.

### **A7. Reviewed\_doc\_hierarchy**

The “reviewed\_doc\_hierarchy” table contains the hierarchy of the original document and the revised document along with reviewer. The composite primary key consists of “source\_doc\_id” and “rev\_doc\_id”. This combination ensures that each version of the reviewed document is kept.

### **A8. Assign\_doc**

Information in this table keeps track of the document and the reviewer. In other words information about who has been assigned which document for review is stored in this table.

### **A9. Assignments**

Information about new assignment goes into this table. It stores the name of the assignment, due date and last review date, and description of assignment.

### **A10. Comments**

The “comments” table stores comments made by reviewer and author. This table is related with ‘users’, ‘documents’, and ‘assignments’ tables to keep track of assignments, documents and comments made by either the author or reviewer. All comments in this table belong to reviewers.

### **A11. Roles**

Roles are defined to assign certain privileges to users who are assigned these roles. The “roles” table keeps information about the roles and users they have been assigned to.

### **A12. User\_role**

The “user\_role” table relates the ‘users’ and ‘roles’ tables. It keeps track of the role assigned to a user.

### **A13. Survey**

Each user is required to evaluate the document they review. To evaluate a document, each user is required to take a mandatory survey and all information is saved in ‘surveys’ table. This information includes the quality of argument, quality of writing, and overall recommendation.

**A14. Evaluation\_by\_author**

The “evaluation\_by\_author” table contains comments made by the author in response to the reviewer’s comments. This table references the ‘users’ and ‘comments’ tables.

**A15. Assignment\_doc**

The “assignment\_doc” table relates the ‘assignments’ and ‘documents’ tables to keep track of which document belong to which assignment.

**A16. Grades**

The “grades” table stores information about the students, assignments, documents and marks assigned. This allows maintaining grades for all students with all other information at the most granular level.