ADAPTING A NOSQL QUERY LANGUAGE FOR QUERYING SQL AND NOSQL DATABASES

BY

Alhamdu Shichet WOJI
B.Sc. COMPUTER SCIENCE (ABU, 2008)
(M.Sc/SCI/7140/2011-2012)

A THESIS SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS
AHMADU BELLO UNIVERSITY,
ZARIA, NIGERIA

JANUARY, 2017

DECLARATION

I declare that the work in this thesis entitled "Adapting a NoSQL Query Language for Querying SQL and NoSQL Databases" has been written by me in the department of Mathematics under the supervision of Prof. S.B Junaidu and Dr. B. I. Ahmad.

The information outlined in the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at any university.

……..……	……	…….
Name of Student	Signature	Date

CERTIFICATION

This Thesis entitled "Adapting a NoSQL Query Language for Querying SQL and NoSQL Databases" by Alhamdu Shichet WOJI (M.Sc/Sci/7140/2011-2012) meets the regulations governing the award of the degree of Master of Science of Ahmadu Bello University, Zaria and is approved for its contribution to knowledge and literary presentation.

_____	_____
Prof Sahalu. B. Junaidu Chairman, Supervisory Committee	Date
_____	_____
Dr. Barroon. I. Ahmad Member, Supervisory Committee	Date
_____	_____
Prof. A. A. Tijjani Head of Department	Date
_____	_____
Prof. Bala Kabir Dean, School of Postgraduate Studies	Date

DEDICATION

I dedicate this work to God Almighty, who has made this work possible. I also dedicate this thesis to my sister Nna Precious Woji.

ACKNOWLEDGEMENT

First and foremost, I would like to thank God for life, love and providence for my life especially in the course of my Masters Degree pursuit.

I am grateful to my supervisors Prof. Sahalu B. Junaidu and Dr. Barroon I. Ahmad for giving their time towards the success this research. Their comments and guidance contributed immensely to the completion of this work.

Words alone cannot express my sincere appreciation to my parents and family; thank you for patience, financial and moral support. To my friends Neyu, Buhari, Philemon, Nathan, Atam, Afiniki, Jabez, Mercy, Edward, Peter, Talatu, Ibrahim, Teku, ThankGod, Femi, Ojuu and many others; I feel indebted to you and I am very grateful for your support.

To my course mates, it has been wonderful studying and collaborating with you. Thank you for your support and advice. I wish you the best in your academic pursuits.

ABSTRACT

Not all kinds of data can find efficient storage and manipulation in Relational (or SQL) database and neither is NoSQL database the best fit for all kinds of data. A hybrid database (a combination of both SQL and NoSQL databases for storage) is a better alternative where structured data are kept in the relational database and the rest in NoSQL database. The hybrid database comes with its challenges; among them is the necessity for the database administrator to learn the query languages of the databases that constitute the hybrid database. This research is focused on using one query language to query the hybrid database via a software layer. NoSQL (MongoDB) query language is adopted as the query language for the hybrid database in this research because it is the fastest growing query language and it is less vulnerable to injections as compared to SQL. The scope of operations supported by the software layer is limited to Create, Read, Update and Delete (CRUD) and MongoDB syntax had to be extended to cater for SQL functionalities necessary to execute CRUD operations in relational databases. The software layer was developed using Java Programming Language; it translates the MongoDB query language syntax to SQL for execution. For evaluation, both databases were fed with the same data via the software layer simultaneously and the CRUD operations were tested simultaneously on the hybrid database and the same result sets were obtained for each database; this asserts that the translation and execution were successful.

TABLE OF CONTENTS

DECLARATION ii
CERTIFICATION iii
DEDICATION iv
ACKNOWLEDGEMENT v
ABSTRACT vi
LIST OF FIGURES x
LIST OF TABLE xii
CHAPTER ONE: INTRODUCTION 1
1.1 Background to the Study 1
1.2 Problem Statement 2
1.3 Aim 2
1.4 Objectives 2
1.5 Methodology 3
1.5 Structure of the Dissertation 4
CHAPTER TWO: LITERATURE REVIEW 5
2.1 Introduction 5
2.2 Overview of the Relational Database Management System 5
2.3 Overview of the NoSQL database 7
2.4 What is causing the migration to NoSQL 8
2.4.1 Big Users: 8
2.4.2 The Internet of things: 9
2.4.3 Big data: 9
2.4.4 The Cloud: 9
2.5 Types of NoSQL 10
2.5.1 Key/Value Database 10
2.5.2 Document Database 10
2.5.3 Column Family Database 11
2.5.4 Graph Database 12
2.6 Related Works 13
2.7 Approach 14

2.7.1 Separate Software layer 15
2.7.2 Load SQL data in NoSQL 15
2.7.3 Load NoSQL data in SQL 15
2.8 Selection of NoSQL Database Candidate 16
2.9 Selection of SQL Database Candidate 19
2.10 Why use NoSQL syntax instead of the familiar SQL 19
2.11 Summary 20
CHAPTER THREE: DESIGN OF SQL AND NoSQL QUERY PROGRAM USING NoSQL SYNTAX 21
3.1 Introduction 21
3.2 Conversion of MongoDB syntax to SQL 21
3.2.1 Conversion of MongoDB's 'Find' statement to SQL's 'SELECT' statement 23
3.2.2 Conversion of 'Insert' statment in mongo to SQL's 'INSERT' statement 25
3.2.3 Conversion of 'Update' statement in mongo to SQL's 'UPDATE' statement 26
3.2.4 Conversion of 'Remove' Statement in mongo to SQL's 'DELETE' statement 27
3.3 Extended Mongo Queries 27
3.3.1 Create Table 27
3.3.2 Join 28
3.3.3 Sub query 29
3.4 Syntax for Querying SQL, NoSQL and both 30
3.5 The architecture of the system 31
3.6 Summary 32
CHAPTER FOUR: IMPLEMENTATION AND EVALUATION OF THE SYSTEM 33
4.1 Introduction 33
4.2 Tools and technologies 33
4.2.1 Java 33
4.2.2 NetBeans 33
4.2.3 MongoDB 34
4.2.4 MySQL 34
4.2.5 JDBC 35
4.3 System Interface 35

4.4 Evaluation of System 36
4.4.1 Evaluation of CRUD queries 37
4.4.2 Evaluation of Extended Queries 44
4.5 Summary 49
CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATIONS 50
5.1 Summary and Conclusion 50
5.2 Recommendations 51
REFERENCES 52

LIST OF FIGURES

Figure 2. 1: Popularity among NoSQL Database (Quora, 2013) 17

Figure 2. 2: United Kingdom's demand of NoSQL Jobs (“IT Jobs Watch”, 2016) 18

Figure 3. 1: Some Commands in MongoDB and SQL 22

Figure 3. 2: Architecture of the System 31

Figure 4. 1: The Interface for the system 36

Figure 4. 2: SELECT ALL query executed on both MongoDB and MySQL 37

Figure 4. 3: FIND (Select All) Executed on MongoDB alone 38

Figure 4. 4: SELECT ALL (Find) Executed on MySQL alone 38

Figure 4. 5: Insert Query Executed on both Databases (*) 39

Figure 4. 6: Confirmation of the Success of the Insert Query (*) 39

Figure 4. 7: SELECT (Find) Query with a Criteria Executed on both Databases 40

Figure 4. 8: Update Query Executed on the Instances of the Criteria in Figure 4.7 41

Figure 4. 9: Confirmation of the Success of the Update Operation in Figure 4.8 41

Figure 4. 10: Another SELECT (Find) query with a Criteria that have been Executed on both Databases 42

Figure 4. 11: Delete (Remove) Query executed on the Instances of the Criteria in Figure 4.10 .. 43

Figure 4. 12: Confirmation of the Success of the Delete Query in Figure 4.11 43

Figure 4. 13: Another Confirmation for the success of the Update (Figure 4.8) and Delete (Figure 4.11) Queries 45

Figure 4. 14: CREATE TABLE query executed on MySQL (@) 45

Figure 4. 15: Confirmation for the Success of the CREATE TABLE Query 46

Figure 4. 16: Sub-query executed on MySQL 47

Figure 4. 17: coreg table Showing the Registration of MATH105 course 48

Figure 4. 18: JOIN query executed on three relational (MySQL) tables..... 49

LIST OF TABLE

Table 2.1 Ranking of Database Enginess (JAX Editorial Team, 2014) 17

Table 2.2: Database and Intelligence Jobs (“IT Jobs Watch”, 2016) 18

Table 3. 1: SQL and MongoDB Terminologies..... 21

Table 3. 2: Read Syntax for SQL and MongoDB (Sian R., n.d) 24

Table 3. 3: Comparison Expressions (“Comparision Query Operators”, n.d)..... 25

CHAPTER ONE: INTRODUCTION

1.1 Background to the Study

Relational database has dominated the database market and had catered for organizational data management needs for decades. It was almost the default choice for any serious data storage in most enterprises. The contemplation was which relational database is to be adopted for a project and not what kind of database will better suit the data in question. The relational database grew and matured over time that it is still considered dependable and reliable (Sadalage *et al*, 2012).

Lately, enterprises have acquired vast volumes of data to store and make available at a very high operation rate to a heavy traffic of users posing a demand that makes the relational database inefficient. This demand can be attributed to the advent of the Web 2.0 that gave rise to very busy websites like social network, e-commerce and the likes. The relational database cannot meet the demands of the new trend. Nance (2013) stated that "relational databases are not well suited for modern web applications that can support millions of concurrent users by spreading the load across a collection of application servers". This challenge made the category of database called Not Only SQL (NoSQL) to emerge and was easily adopted to cater for the prevalent data storage needs.

The NoSQL database found express acceptance but could not become a replacement of the relational database. They have some weaknesses that make the relational database outstanding by the solutions they proffered. Nance *et al.,* (2013) explained that NoSQL databases can be used with applications that have: large transaction volumes, have the need for low-latency access to massive datasets, and have the need for nearly perfect service availability while operating in an

unreliable environment. This implies that both types of database (relational and NoSQL) have their strengths and weakness; none is a perfect fit for all kinds of data.

There are situations where an enterprise has part of its data suitable for relational database storage and the other part that will fit well into the NoSQL database. Forcing the entire data into a single database will not serve the data management needs efficiently. It can only be better when the data are segregated according to the data model that best suits their usage and are stored in the database system that supports the data models. The application (software) of the enterprise will have to interact with more than a single database which Jafarpour *et al.,* (2015) have termed it as hybrid database.

1.2 Problem Statement

The hybrid database has its demerits. The most crucial problem to this research is that the hybrid database requires the programmer(s) to learn the query languages for each database system that make up the hybrid database; for the programmer to be able to access and manipulate the databases that make up the hybrid database.

1.3 Aim

This research is aimed at bridging the gap between the two data stores by providing a platform for querying SQL and NoSQL databases using NoSQL query language.

1.4 Objectives

The objectives of this research are to:

a. Create a software layer that serves as a common interface for accessing both SQL and NoSQL databases.

b. Design an interpreter to translate NoSQL query syntax to SQL syntax in the software layer.
c. Implement and evaluate the software layer (system).

1.5 Methodology

Below are the procedures that will be followed to carry out this research:

a. Literature of researches that relates to bridging the gap between the SQL (Relational) databases and the NoSQL databases has been reviewed.
b. The selection of a candidate database to represent SQL databases and another candidate database to represent NoSQL databases. Both types of databases (SQL and NoSQL) are made up of several database management systems produced by different vendors. One representative from both categories was adopted for this research.
c. Adoption of the query language of one of the candidate databases query language to be the language for querying the hybrid system via the software layer. Developing a conversion scheme for translating the adopted language to the native query language of the other candidate database (SQL).
d. The design of the architecture of the system (software layer); it served as a framework that guided the implementation of the software layer.
e. The system was tested using the same data set for both databases to ensure the queries that are run on software layer produces the same result from both databases. This confirmed that queries were correctly converted and executed.

1.5 Structure of the Dissertation

Here is the organization of the rest of the dissertation. Chapter 2 covers the overview of the two types of databases (MySQL and NoSQL) and the review of related researches that were conducted in an attempt to bridge the gap between SQL and NoSQL databases as well as the approach to be taken to achieve the aim of the research. Chapter 3 focuses on the design of the of software layer (system). It covers the comparison of both query syntaxes and how to map out a conversion scheme. Chapter 4 has the implementation and evaluation of the system. This includes the technology and tools that are used to implement the system and the result obtained in the course of testing the system. Chapter 5 has the conclusion and the possible future research that can extend this research.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

This chapter will give insights about the Relational Database and the NoSQL database technologies. The previous researches conducted in an attempt to bridge the gap between different database technologies will be discussed and used as guides to the solution this research will proffer. The criteria for selecting the candidate databases to represent Relational and NoSQL databases are covered in this chapter.

2.2 Overview of the Relational Database Management System

The relational database stores data in tables. The table is a two dimensional grid of rows and columns. Each table is used to store a data object (or entity) like student, lecturer, course, course registration. Each column of the table stores a single attribute of the object (attributes of student include: name, admission number and address). The row contains the information about an instance of student; implying each row contains data about a single/unique student. To store data in a relational database, a table has to be created; specifying the attributes (or columns) and the data type to be stored in the columns (such as integer, character and string); the individual data for each row must conform to the data type of the columns which the data will be placed inside. That is why it called a schema database or a structured database. The data that fits into the relational database are called structured data.

Each row of data in a relational database table must be unique. This is achieved by ensuring that one or more of the attributes of a table must have unique values for each row of data. The attribute(s) use to achieve this uniqueness is (are) called Primary Key(s).

A relational database can have more than a table, and relationships can be created among these tables. To create a relationship among any two tables, there must be an attribute (column) in the second table that matches the primary key in the main table. The matching attribute in the second table is called the Foreign key. The concept of relationships makes the update operations automatic without requiring the user to effect the update to several records that reference the updated data (‘SQL - RDBMS Concepts’, 2016).

Enterprise applications tend to have many people looking at the same body of data at once, possibly modifying that data. As a result, we have to worry about coordinating these interactions to avoid concurrency such things as double booking of hotel rooms. Concurrency is notoriously difficult to get right, with all sorts of errors that can trap even the most careful programmers. Relational databases have handled this by controlling all access to their data through transactions (Sadalage *et al*., 2012).

Any operation that propagates any change (e.g Insert, Update and Delete) to the relational database is called a transaction. A transaction has the following four standard properties called ACID (an acronym for Atomicity, Consistency, Isolation and Durable). Atomicity means the pieces of information tied to a transaction must all be committed or none will be committed. Consistency means a transaction creates a new state for the database, if the transition to the new state fails; the state of the database is reversed to the former, which is the state of the database before the transaction was initiated. Isolation means a transaction in process remains isolated from other transactions until it is committed. Durable implies committed transactions must be stored to a permanent memory (‘SQL-Transaction’, 2016).

The relational database uses the Structured Query Language (SQL) to create and manipulate the contents of the database. Data can be stored only when a schema has been defined. SQL is used

to define the schema, modify the schema, insert new data, delete data and modify the data in the database.

The relational database is scaled up when its capacity is being stretched and due to increasing traffic to the database. It is achieved by adding to the number of the processors and the memory capacity. It is an expensive venture, because it is can be achieved using costly servers. Cattel (2012) stated that the scaling of the relational database has a limit (roof) to the number of processors and memory that can be added. If the volume of data keeps increasing as well as the traffic of users, the scaled up relational database will not be able to meet the demands of clients in a short while.

2.3 **Overview of the NoSQL database**

The advent of web 2.0 that birthed the social network sites among others which requires a database server that could make available an enormous volume of data to a heavy traffic of users. The relational databases were scaled up and tuned up but it still fell short of the demands. Then arose alternative databases that met the demands and they had other data models other than the relational data model.

Cattell (2010) listed out the common features of the NoSQL data stores as follows:

a. They have the ability to scale horizontally over many nodes.
b. They can easily replicate and partition data over multiple nodes.
c. Their concurrency model is often weaker then ACID.
d. The data schema is flexible enough to add new attributes to the data.

e. Many NoSQL data stores have inbuilt support for automatic sharding (the fragmentation of a data set and storing the fragments in different nodes of a cluster) unlike manual sharding in case of traditional RDBMS.
f. Several features of SQL such as join and global transactions are not supported by NoSQL data stores.
g. NoSQL data stores have sparse data model unlike RDBMS.

Sadalage (2014) also observed that the common features among the NoSQL databases include:

a. Not using the relational model
b. Running well on clusters
c. Mostly open-source
d. Built for the 21st century web estates

2.4 What is causing the migration to NoSQL

There are four major and interrelated reasons that promote the migration to NoSQL database (Couchbase, n.d). They include:

2.4.1 Big Users: the amount of time people spend online has always been increasing which has tremendously increased the number of concurrent user thereby making a norm for some online applications to have millions of users per day. A better database scalability technology is required to cater for the large number of users combined with the dynamic nature of usage patterns. With relational technologies, many application developers find it difficult, or even impossible, to get the dynamic scalability and the level of scale they need and still maintain the performance that users' demand. Many are turning to NoSQL for help.

2.4.2 The Internet of things: The amount of machine-generated data is increasing with the explosion of digital telemetry. This is the use of sensors to collect environmental data such as location, weather data, road traffic and many others. They are in factories, farms, hospitals, Warehouses, home appliances, cars, mobile phones and tablets (things). Innovative enterprises leverage on these data to enhance informed decisions like reduce cost, eliminate waste, increase efficiency and customer satisfaction; these increase business agility. However, telemetry data are semi-structured and continuous making the relational database an unsuitable storage for them.

2.4.3 Big data: huge volumes of data are being obtained from third parties like social networks carrying Personal user information, geo-location data, social graphs, user-generated content, machine logging data, and sensor-generated data. Storing and processing these data has changed the nature of communication, shopping, advertising, entertainment, and relationship management. Applications that don't find ways to leverage on it quickly fall behind.

2.4.4 The Cloud: business applications have moved from the single-user applications that ran on a local PC to the two-tier architecture (Server-Client) which supports a limited number of users and now to the prevailing three tier (internet) architecture that run in the cloud where applications are accessed via web browsers and mobile apps. It supports an unlimited number of users. The cloud requires a good scaling capability to balance the load of serving their numerous users.

2.5 Types of NoSQL

Beyond these similarities among the NoSQL databases, they differ in their data models, architecture and fault tolerance. Abramova *et al.,* (2014) categorized the NoSQL databases according to their data models as follows:

a. Key Value stores
b. Document stores
c. Columnar stores
d. Graph stores

2.5.1 Key/Value Database

In this database, data is saved as a pair of key and value. The key is the attribute and value is the data/value for the key. It is not a table but can be visualized as a two columns table with ID and NAME columns; ID storing the key and NAME storing the values. The operations include: get (view the value of a key), put (insert a value for a key) or delete (remove the value of a key). When inserting a pair whose key already exist in the database, the former is replaced. The key is a string and the value is stored as a blob not caring what is inside; it is up to the application to find out what data type it is. They are extremely useful in building specialized application with super fast query capabilities. Key value is not a good store for data that have relationship and may be queried together. Examples of Key-Value database include: Riak, Redis, Memcached, BerkeleyDB, DynamoDB and Couchbase.

2.5.2 Document Database

Documents are the main concept in document databases. The database stores and retrieves documents; which can be XML, JSON, BSON, and other formats. These documents are in

hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents as a collection of key-value pairs where a key (attribute) may have embedded attributes (sub-attributes) with their values; this gives rise to a nested document. For example, the attribute address has sub attributes like: number, street name, district, and town.

The Document Database is ideal for content management systems, blogging platforms, web analytics, real-time analytics, and ecommerce-applications. It is not a good store for applications that require complex queries especially those spanning different units of storage within the database (Sadalage, 2014). The examples of document databases include: MongoDB, CouchDB, Terrastore, OrientDB and RavenDB.

2.5.3 Column Family Database

Column-family databases store data in column families as rows that have many columns associated with a row key. Column families are groups of related data in columns that are often accessed together. In shop instance, customer profile will be in the one column family, customer order will be in another column family.

Each column family can be compared to a container of rows in a relational table where the key identifies the row and the row consists of multiple columns. Unlike the traditional defined schemas of relational databases, column-based NoSQL solutions do not require a pre-structured table to work with the data. Each record comes with one or more columns containing the information and the columns of each record can be different, and columns can be added to any row at any time without having to add it to other rows.

Column-based databases can be thought of as a two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and un-structured data to be stored and used.

When a column consists of a map of columns, then we have a super column. Think of a super column as a column that houses several other columns. Example of the column family database include: Cassandra, HBase and Hypertable (Sadalage, 2014).

2.5.4 Graph Database

Graph databases allow you to store entities and relationships between these entities. The entities are nodes that have properties and relationships are the edges that can have properties too. Edges have directional significance. The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships. The value of graph databases is derived from the relationships. Relationships have a type, a start node, an end node, and can have properties of their own. The properties on the relationships can be used for inference and can be used to query the graph.

Since most of the power from the graph databases comes from the relationships and their properties, a lot of thought and design work is needed to model the relationships in the domain that we are trying to work with. Changing existing nodes and their relationships is similar to data migration, because these changes will have to be done on each node and each relationship in the existing data. Examples of Graph database includes: Neo4J, Infinite Graph, OrientDB and FlockDB (Sadalage, 2014).

2.6 Related Works

Researches that involve the both SQL and NoSQL database can be looked at from two perspectives; Migration from SQL to NoSQL and Polyglot Persistence (where an application uses more than one database to keep its data). This research is of the latter. In this section, an overview of researches conducted to harness the SQL and NoSQL are discussed.

Varley (2009) discussed the variations of the NoSQL databases from the SQL databases are discussed. This includes scalability, performance, simplicity and the differences in their conceptual modeling techniques. The advantages and limitations of the categories of the NoSQL databases were presented too in the research document. The research did not address the issue of unified retrieval of data from both databases.

Artem *et al*., (2009) attempted to bridge the gap between the semantic web and relational database. Semantic web uses the RDF as the language for representing data. As several researches had attempted to store and query RDF data in a relational database, in their (Artem et al) research they opted for the translation of SPAQL (RDF query language) to SQL to bridge the gap. This was achieved by mapping technique. Mapping is also a technique that will be considered in translating NoSQL syntax to the SQL equivalent.

Ferreira (2012) proposed the use of SQL database over a NoSQL database. Considering the blessings that came with NoSQL fast writes and reads, very large scale distributed database, flexible schemas and the downside of not having a standard query language as SQL; the decision was to forego the compromises of migrating data from SQL to NoSQL. The approach adopted for the work used an (open source) SQL engine (as front end) over a NoSQL-Cassandra backend

(Ferreira, 2012). The storage system here marries the two; the SQL and the NoSQL technologies but stored data in only one data store.

Roijackers (2012) tried to query both SQL and NoSQL databases using SQL syntax. This work focused on getting a unified data from both databases. The approach here is to load virtual data from the NoSQL in the SQL database such that when an SQL query requiring data from the NoSQL data, a mechanism is put in place to draw up the NoSQL data during query execution. Since the NoSQL comes in varying data models, this research used a triple format because any data model can be converted and represented in the triple format. The NoSQL data is converted to the triple format and stored as virtual data in the SQL database. The SQL syntax was extended with SPARQL to enable the fetching of the triple formatted data. The procedure was expensive and time wasting due to series of Join operations and is susceptible to data loss. This work has exposed the need for syntax extension to enable a query language to query a non-native database.

Another approach by Adeyi (2014) to bridging the gap between SQL database and NoSQL database was by building a software layer over both databases that enables the access of both databases with a single query. They single query statement in this research used the ‘and’ conjuction to aggregate the query languages of each of the databases that make up hybrid database. This still requires the user to learn both query languages to form the aggregated query.

2.7 Approach

From the related works above, it will be observed that marrying different database technologies is a great idea but these varying technologies do not work together seamlessly. Roijackers (2012) pointed out that retrieving data from multiple sources implies learning to work with different

technologies, learning more query languages and these makes work quite cumbersome for the user. Roijackers (2012) suggested three ways to approach these challenges, they include:

2.7.1 **Separate Software layer:** this approach suggests that an abstraction layer is build above both the SQL and NoSQL databases. With one query language data could then be retrieved from both data sources through this abstraction layer. The abstraction layer is responsible for retrieving the relevant data from the underlying SQL and NoSQL databases.

2.7.2 **Load SQL data in NoSQL**, either materialized or virtual: this would imply that we use the NoSQL database as the primary database. The SQL data is virtually loaded into the NoSQL database as the primary database. The SQL data is virtually loaded into the NoSQL database and can be read using the NoSQL query language. This may require an extension to the NoSQL database to add support for accessing the SQL data. Depending on the specific SQL and NoSQL solutions, transforming structured SQL data to fit in an unstructured NoSQL database (a less constrained environment) should be relatively easy.

2.7.3 **Load NoSQL data in SQL**, either materialized or virtual: Another approach is to incorporate the NoSQL data in the relational database. This incorporation can be achieved by materializing the NoSQL data in the relational database or by making the NoSQL data virtually available and thus does not actually retrieve the NoSQL data until query execution. The developer needs to open a connection to the SQL database, and can also read the NoSQL data. The most obvious challenge with this approach is how to represent the unstructured NoSQL data in an SQL

database, and thereafter allow the developer to easily construct queries that read data from the NoSQL data source.

From these approaches, the first is quite feasible and flexible. It conforms to Adeyi's (2014) research where the procedure is less expensive and there was no data lost or data degradation because no data transformation is carried out during retrievals. This research adopted the first approach.

2.8 Selection of NoSQL Database Candidate

The NoSQL databases as mentioned are of four major categories. There are rankings (JAX Editorial Team, 2014) that have place MongoDB to be the most popularly used and referenced. The ranking is summarized in table 2.1. In October 2013, only MongoDB made the top ten among NoSQL databases and it was 6^{th} position. In September 2014, MongoDB was 5^{th} position, Cassandra 8^{th} and Redis 10^{th} positions in the ranking. As at January 2016, MongoDB assumed the 4^{th} position, while Cassandra and Redis maintained their 8^{th} and 10^{th} positions in the ranking. As at 2014, JAX Editorial team established that MongoDB is the youngest of the most popular database engines implying its popularity is growing very fast.

Also, Quora (2013) claimed that MongoDB is the most popular NoSQL database. In figure 2.1, MongoDB's sector is the largest sector among other sectors representing other NoSQL databases.

Table 2.1: *Ranking of Database Enginess (JAX Editorial Team, 2014)*

RANK	DB ENGINE	YEAR INITIAL RELEASED
1	Oracle 12	1980
2	MySQL	1995
3	Microsoft SQL Server	1989
4	PostgreSQL	1989
5	MongoDB	2009
6	DB2	1983
7	Microsoft Access	1992
8	SQLite	2000
9	Cassandra	2008
10	Sybase ASE	1987

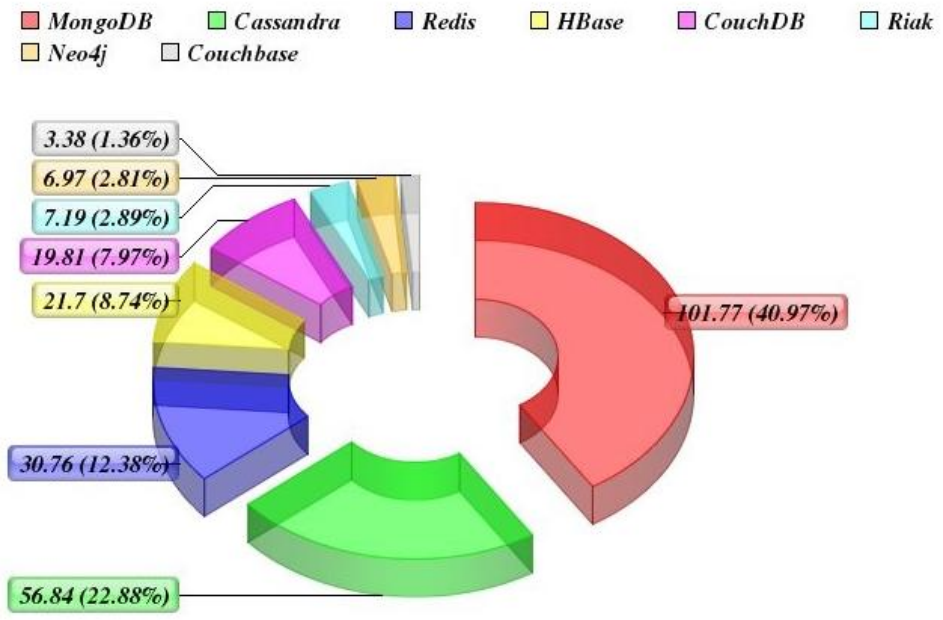


Figure 2. 1: *Popularity among NoSQL Database (Quora, 2013)*

In the United Kingdom (U.K.), the demand of jobs requiring NoSQL skills is skyrocketing as shown in figure 2.2 as pointed out by ("IT Jobs Watch", 2016).

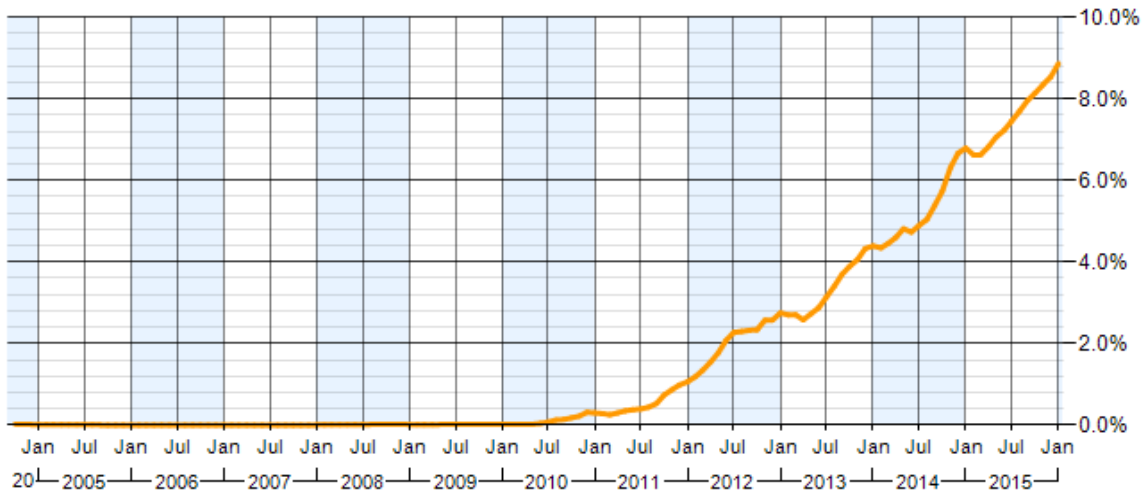


Figure 2. 2: *United Kingdom's demand of NoSQL Jobs ("IT Jobs Watch", 2016)*

IT Jobs Watch (2016) grouped IT skills into categories. The 'database and business intelligence category' is given in the table 2.2. The figures indicate the number of jobs and their percentage against the total number of IT jobs in that category. Up to 20 skills are shown per category.

Table 2.2: *Database and Intelligence Jobs ("IT Jobs Watch", 2016)*

RANK	PROPORTION	NAME	RANK	PROPORTION	NAME
1	1900 (34.25%)	MongoDB	**11**	311 (5.61%)	PostgreSQL
2	1535 (27.67%)	SQL Server	**12**	270 (4.87%)	Data Warehouse
3	1242 (22.39%)	Big Data	**13**	257 (4.63%)	Apache Hive
4	1065 (19.20%)	Hadoop	**14**	186 (3.35%)	CouchDB
5	1019 (18.37%)	MySQL	**15**	184 (3.32%)	Neo4j
6	882 (15.90%)	Apache Cassandra	**16**	174 (3.14%)	Couchbase
7	794 (14.31%)	SQL Server 2014	**17**	158 (2.85%)	HBase
8	576 (10.38%)	Relational Database	**18**	146 (2.63%)	Riak
9	337 (6.08%)	Redis	**19**	138 (2.49%)	Amazon DynamoDB
10	327 (5.90%)	RDBMS	**20**	123 (2.22%)	SQL Server 2008

Table 2.2 indicates that MongoDB skill is the most demanded job skill among database jobs in the United Kingdom and by way of inference, it can be called the most popularly used NoSQL database. MongoDB is the selected database to represent the NoSQL databases in this research.

2.9 Selection of SQL Database Candidate

From the tables 2.1 and 2.2, it will be observed that the most popular of the SQL databases include: SQL Server, Oracle, MySQL, PostgreSQL. The most popular that is open source amongst them is MySQL. MySQL has been adopted to represent SQL databases in this research.

2.10 Why use NoSQL syntax instead of the familiar SQL

Considering the age of the MongoDB and its popularity, it will be logical to predict that in the near future it will be the most popular or the dominant database if the trends of database usage and technology does not witness a major turnaround from what is obtainable in table 3.1 and 3.2 and in figure 3.1. So the Mongo query language will soon become as familiar as or even more familiar than the SQL.

Ron *et al*., (2015) stated that: "the breaking of query structure that has been done in SQL injection is harder to do with a JSON structured query". In their work they mentioned that Queries and Data that are represented in JSON format are better than SQL in terms of security because they are better defined, very simple to encode/decode and also has good native implementations in every programming language. MongoDB have both the data structure and query in JSON/BSON format.

MongoDB syntax is the adopted language for this research because:

a. it is the most demanded job skill and has the prospect of becoming more popular than SQL considering its growth rate from tables 3.1 and 3.2, and figure 3.1
b. MongoDB syntax is less vulnerable to query injections.

2.11 Summary

This chapter gives an overview of both relational and NoSQL databases, types of NoSQL databases and provides explanation of the factors that causes the migration from SQL to NoSQL database. Related researches are also reviewed in which some served as a guide to some of the methods that will be used in this research. The selection of the candidate database for both SQL and NoSQL databases that make up the hybrid database in this research. The reasons for adopting NoSQL syntax as the query language for the hybrid database in this research has been discussed in this chapter.

CHAPTER THREE: DESIGN OF SQL AND NoSQL QUERY PROGRAM USING NoSQL SYNTAX

3.1 Introduction

This chapter discusses the scheme for translating NoSQL syntax to SQL which will be the guide for implementing the interpreter in the software layer. The architecture of the system for querying the hybrid database using NoSQL syntax is shown and discussed in this same chapter.

3.2 Conversion of MongoDB syntax to SQL

One of the crucial tasks to undertake in this research is the conversion of the Mongo syntax to SQL; since Mongo syntax is the adopted lingua franca for both databases. Mongo also have quite expressive and numerous command (queries) even though it is not as old/matured as the SQL.

Table 3. 1: *SQL and MongoDB Terminologies*

SQL	**MONGODB**
Database	Database
Table	Collection
Row/Record	Document
Column	Field

The table 3.1 shows the SQL terms and their Mongo equivalents. SQL organizes its data in tabular structures while MongoDB organizes its data in a structure called collection. Distinct records are stored as rows of data in SQL while MongoDB has it in a JSON/BSON document form. SQL has its attributes as column names of the table while Mongo has its attributes as the fields of the JSON/BSON document.

Due to the difference in their data models, not all MongoDB query commands have an SQL equivalent likewise, not all SQL commands have a Mongo syntax equivalent. They have a number of commands in common as depicted by the figure 3.1.

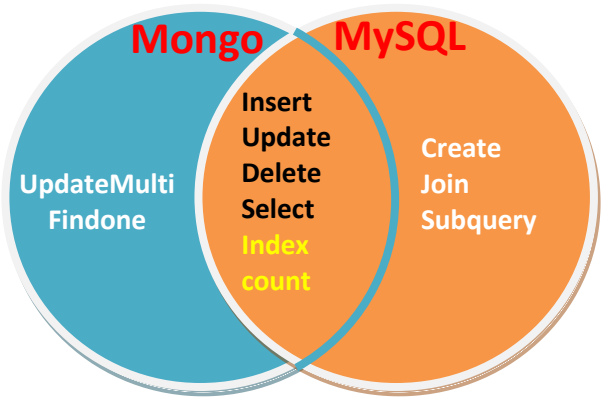


Figure 3. 1: *Some Commands in MongoDB and SQL*

The commands/operations that both databases have in common that will be used in this research are termed CRUD (Create, Read/Retrieve, Update and Delete) operations (Polo et al, 2001). The Create in CRUD means to create a new records/documents, it is achieved using the Insert statement. The Read or Retrieve means to view records/documents that have been stored; it is achieved using SELECT (for SQL) and find (for MongoDB). The Update is use to modify record(s)/document(s) that are already in a table/collection. The Delete is used to remove records/documents that are in a table/document.

The Relational database requires the user to create a database, and then create a table or tables in the created database before data/records can be saved in the database. To create a table, it requires that the user to state the field names, the data type for each field and the maximum length of character each field can accommodate. Unless the table is created the insert, update, delete and select operations cannot be executed.

Unlike the SQL database, MongoDB always has a default database created but the user can create more. Instead of tables, its structure is called Collection. The user may create a collection with the create collection command; the collection will remain invisible to the user until data/document have been saved in the collection. The insert command in mongo performs two (2) functions: it creates a collection (if the collection name in which the document is to be inserted does not exist) before it inserts the document or it only inserts the document if the collection name already exist in the database.

SQL have operations like CREATE TABLE, JOIN operation and SUBQUERY that are peculiar to only relational database. These operations/commands are required for some of the CRUD commands to be executed in relational database. Without CREATE TABLE in SQL none of the CRUD commands can be executed. The JOIN and the SUBQUERY commands enhance the SELECT command in SQL.

For mongo syntax to be used as the adopted lingua franca for both databases, the mongo syntax has to be extended in the JSON format to incorporate SQL commands that are not native to MongoDB and are essential for the execution of queries in this research. These SQL commands include: create table, join and sub query queries.

3.2.1 Conversion of MongoDB's 'Find' statement to SQL's 'SELECT' statement

To read/view data in a relational table the user will use a SELECT command.

SQL: SELECT *projections* FROM *tablename* WHERE *criteria/condition*

Mongo: db.*tablename*.find({*condition/criteria*},{*projections*})

The table 3.2 has examples of the SQL **SELECT** statements and their Mongo **find** equivalents.

In Table 3.2, the MonogDB find syntax has a parenthesis, inside which there can be one, two or no curly brackets. The first curly is for criteria (for 'where conditions') and the second is the fields projection. If it is only one curly bracket then only the where condition will be specified in it. If the query is to project fields without using any where condition, then there must be two curly brackets with the first one empty and the second specifying the projections. When the find parenthesis has no curly brackets, it implies they user wants to view the whole of the collection.

Table 3. 2: *Read Syntax for SQL and MongoDB (Sian R., n.d)*

SQL SELECT	**MONGO FIND**
SELECT * FROM user	db.user.find()
SELECT * FROM user WHERE name= 'Bala'	db.user.find({name:Bala})
SELECT * FROM user WHERE name= 'Bala' AND age=25	db.user.find({name:Bala,age:25})
SELECT * FROM user WHERE name= 'Bala' OR age=25	db.user.find({$or:[{name:Bala},{age:25}]})
SELECT name, age, regno FROM user WHERE name= 'Bala' OR age<25	db.user.find({$or:[{name:Bala},{age:{$lt:25}}]},{name:1, age:1,regno:1})
SELECT name, age, regno FROM user	db.user.find({},{name:1,age:1,regno:1})

To specify the criteria, each criterion is defined by a pair of field and value with a colon between them and each pair is separated from the next with a comma, for example

find({age:18,level:100}). For any field that is to be projected, the field name is assigned the value 1 in the second curly bracket. If the value for the field is 0, then the field will not be projected for example find({age:30},{name:1,age:1,address:0}) will display only the name and age attributes.

Comparison expressions that can be included in the criteria section in the MongoDB syntax are explained in table 3.3 with examples.

Table 3. 3: *Comparison Expressions ("Comparision Query Operators", n.d)*

SQL	**Mongo**	**Example**
age<20	age:{$lt:20}	db.user.find({age:{$lt:20}})
age<=20	age:{$lte:20}	db.user.find({age:{$lte:20}})
age>20	age:{$gt:20}	db.user.find({age:{$gt:20}})
age>=20	age:{$gte:20}	db.user.find({age:{$gte:20}})
age>20 and age<25	age:{$gt:20,$lt:25}	db.user.find({age:{$gt:20,$lt:25}})

3.2.2 Conversion of 'Insert' statment in mongo to SQL's 'INSERT' statement

To save a record in a relational table called user, the command to use is:

INSERT INTO user (admno, name, age, state) VALUES ('U01', 'Bala', 22, 'Kaduna')

The mongo equivalent of SQL query is:

db.user.insert({admno:U01,name:Bala,age:22,state:Kaduna})

SQL starts by arranging the fields first and then arrange the values in the order of the corresponding fields. But mongo arranges each pair of field and value with a column between them with a comma separating the pairs.

The conversion process will collect the fieldnames together as a collection and the values together as another collection. In the construction of the SQL insert statement value set has to be arranged in an order corresponding to the arrangement of the field set.

3.2.3 Conversion of 'Update' statement in mongo to SQL's 'UPDATE' statement

UPDATE command is use to modify a record in a table. For example, to modify the 'user' table using SQL:

UPDATE user SET age=25 WHERE name='Bala'

The mongo equivalent is

db.user.update({name:Bala},{$set:{age:25}})

db.user.update({condition},{$set:{fields : new values for the fields}})

To update more than one field in a document, the mongo syntax will look this:

db.user.update({name:Bala},{$set:{age:25,state:Kano}}). The record/document having Bala as value for name will have the field 'age' set to 25 and the field 'state' set to Kano.

Though in mongodb, the update command is two types:

a. db.user.updateone() – updates only the first document that meets the criteria (WHERE condition)
b. db.user.update() - updates all the documents that meets the criteria (WHERE condition)

In this research, only *db.user.update()*is used for update operation. In converting the update syntax, the new values to be set for the fields are collected in the second curly bracket and the

WHERE condition in the first. These sets of data will be used to construct the SQL update syntax.

3.2.4 Conversion of 'Remove' Statement in mongo to SQL's 'DELETE' statement

SQL calls it delete while Mongo calls it remove.

To delete a record from user table, the syntax is:

DELETE FROM user WHERE name='Bala'

The mongo equivalent is:

db.user.remove({name:Bala})

There is only one curly bracket within the parenthesis; which is the curly bracket for criteria. To convert to SQL, it is only the WHERE condition (criteria) and the table name that is required to construct the delete syntax in SQL.

3.3 Extended Mongo Queries

As earlier mentioned, the extended queries are queries essential queries in SQL that do not have an equivalent in the MongoDB syntax due to the differences in their data models. MongoDB syntax have been extended to cater for the following SQL queries: Create table query, Join query and Sub-query.

3.3.1 Create Table

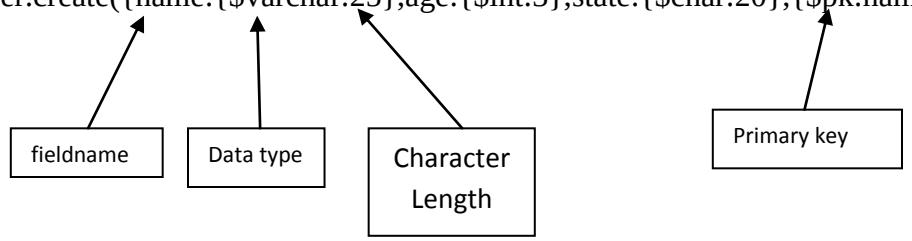
MongoDB does not require the fieldnames (column names) and their data type and character length to be specified in order to create a collection as it is a requirement for creating a table in SQL. The mongo has to be extended in this research using the JSON format so that using the

mongo syntax the create table operation can be achieved in the relational database via the software layer above it.

It is from this query statement: db.user.find({age:{$gt:20},height:{$lt:6}}) the create table syntax is formed.

db.tablename.create({col1:{datatype:xterlength},col2:{datatype:xterlength},{$pk:col}})

db.user.create({name:{$varchar:25},age:{$int:3},state:{$char:20},{$pk:name}})



The above extended mongo syntax for create table will produce:

CREATE TABLE user (name VARCHAR (25), age INT (3), state CHAR (20), PRIMARY KEY(name));

In this research, the data types for creating a relational table are limited to only four. They are: INT, FLOAT, CHAR and VARCHAR which are represented by this research in the mongo syntax as $int, $float, $char and $varchar respectively.

3.3.2 Join

Join operation in SQL produces a combination of data from more than one table with a condition specifying a relationship between the tables. The condition normally specifies data set in a particular field (column) of one table to the data set of a field in another table. Format of the join query looks like:

SELECT * FROM table1 JOIN table2 WHERE table1.columnnA =table2.columnB

SELECT * FROM table1 JOIN table2 WHERE table1.columnnA=table2.columnB and table1.columnA<50

The extended mongo syntax to carry out the SQL 'join' commands above is:

db.table1+table2.find({$join:[{table1:column,table2:column}]})

Below are two other extended mongo syntax commands to perform the SQL join commands.

db.table1+table2+table3.find({$join:[{table1:column,table2:column},{table2:column,table3:column}]})

db.student+coreg+course.find({$join:[{student:admno,coreg:admno},{coreg:code,course:code}],course+title:Calculus},{student+lname:1,student+fname:1,coreg+code:1,course+title:1,course+semester:1})

The above mongo syntax for 'join' operation is derived from the MongoDB find syntax with an OR criteria as below:

db.user.find({$or:[{name:Bala},{age:50}]})

3.3.3 Sub query

In SQL, the where condition for criteria can be another query. Such a query is called a sub-query.

SELECT * FROM user WHERE city IN (SELECT city FROM address WHERE country='Nigeria');

The query above shows the records in the user table whose city attribute is used to verify in the address table if that city is in Nigerian. The query will show records of users who are living in Nigerian cities (or rather in cities that are in Nigeria).

The extended mongo syntax for the subquery is also derived from the MongoDB find syntax that has an OR criteria such as: db.user.find({$or:[{name:Bala},{age:50}]})

Below is the format of the extended sub query and under it is an example

db.tableA.find({columnA:{$sub:[{tableB:{columnB:valueB}}]}})

for example: db.user.find({city:{$sub:[{address:{country:Nigeria}}]}})

The query gets the data from columnA in TableB whose columnB counterpart is valueB and uses the data as criteria to fetch records from TableA. Similarly, the query gets citynames from the column *city* in the table (*address)* whose *country* column has the value Nigeria, it is the citynames obtained from *address* that is used as criteria for querying in table (*user*).

3.4 Syntax for Querying SQL, NoSQL and both

In this research, all the queries will be in MongoDB syntax. To be able to query the databases differently, the user can query:

a. Only MongoDB using the MongoDB syntax. For example: *db.user.find()*
b. Only the relational database using the MongoDB syntax with the @ prefix . For example: *@db.user.find()*
c. Both MongoDB and relational database simultaneously using the mongo syntax with the * prefix. For example: **db.user.find()*

The extended Mongo queries are applicable to only the relational database. So these queries will use @ prefix.

3.5 The architecture of the system

Figure 3.4 is the architecture of the system of this research. The whole system is divided into three (3) major components, they are: the interface, the software layer and the backend (MySQL and MongoDB).

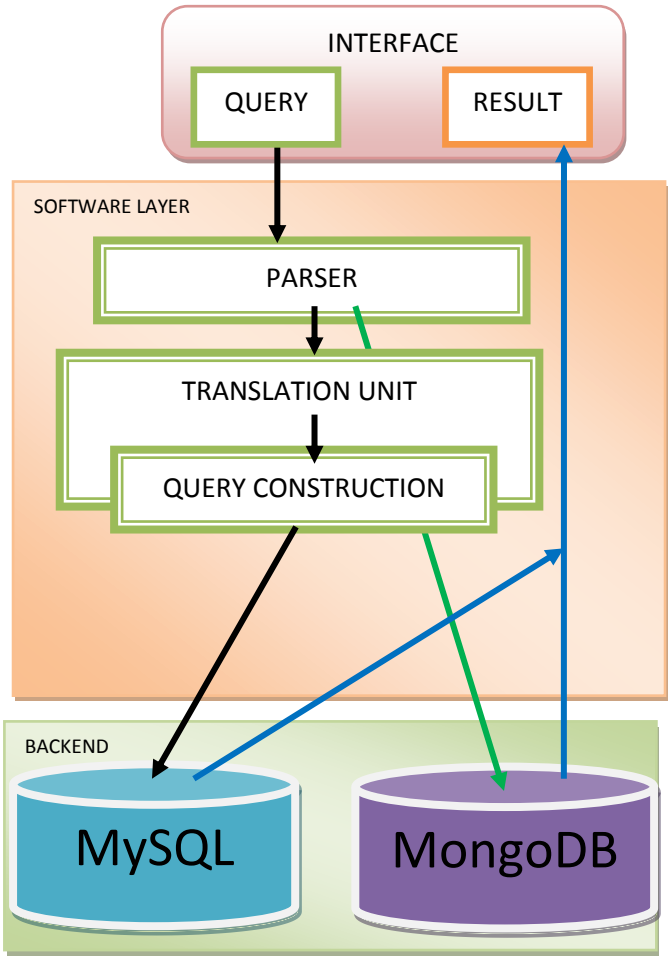


Figure 3. 2: *Architecture of the System*

The interface collects user query statements and displays results of the queries. When a query is sent, the parser checks if it is for the SQL database; it breaks the query into small tokens to enable the translation unit convert these Mongo tokens to the SQL conditions and projections units. If the query is only for MongoDB, the query will not be converted it SQL; it will be sent to MongoDB for execution. The output from the translation unit is used to construct the equivalent

SQL query in the construction unit which is an extension of the translation unit. The constructed query is sent to the database for execution. The result of the query is retrieved and sent to the interface for display. Unlike the architecture in Adeyi (2014), the software layer in this research performs translations of Mongo syntax to SQL.

3.6 Summary

In this chapter, MySQL was chosen to represent relational database and MongoDB to represent NoSQL database. MongoDB syntax is the query language that has been adopted for this research and a theoretical scheme to convert MongoDB syntax to SQL has been produced. The adopted query language has been theoretically extended to enable it carry out some functions in the relational database that does not exist in MongoDB. The @ and * symbols are the prefixes use in the MongoDB syntax to determine which of databases in the backend to query. Lastly, the architecture of the proposed system has been discussed.

CHAPTER FOUR: IMPLEMENTATION AND EVALUATION OF THE SYSTEM

4.1 Introduction

In this chapter, the outline of the technology and tools used to implement the system are briefly discussed. The images of the interface with queries and results are also presented.

4.2 Tools and technologies

The tools and technologies used to implement this system includes: Java Programming language, Netbeans IDE, MongoDB server, MySQL server, JDBC for MySQL and MongoDB.

4.2.1 Java

Java was originally developed by James Gosling at Sun Microsystems and was released in 1995 as a core component of Sun Microsystems. Sun Microsystems has been acquired by Oracle Corporation in 2010. The Java language syntax was majorly derived from C and C++ syntaxes. Java is one of the most popular general-purpose computer programming languages. It is class-based, object-oriented. It allows application developers to write java codes once and run it on any platform because Java codes can run on all platforms that support Java without the need for recompilation. It is be used to produce standalone application software and is also used to produce web applications. In this research, it is used to produce a standalone application.

4.2.2 NetBeans

NetBeans is a Java-based Integrated Development Environment (IDE). This Java software development environment is written in Java, it also supports other programming languages such as C, C++, PHP, and HTML5. It works on different platforms like Mac OS, Linux, Windows and any platform that support JVM (Java Virtual Machine). NetBeans was bought by Sun

Microsystems and now a property of Oracle. It is open source and is free for commercial and non commercial use (Netbeans, 2016). NetBeans 8.0.1 is the platform use to develop software layer of this research because it is free and user friendly.

4.2.3 MongoDB

MongoDB was developed by 10gen using C++. It was released in 2009. This database is efficient and built for high performance. It is consistent, fault tolerant and persistent. In MongoDB the documents are mainly stored in BSON (Binary JSON) format. BSON is efficient both in storage space and search speed when compared to JSON (Harris, 2013). MongoDB uses GridFS (Grid File System) as a specification for storing large files (Nayak et al, 2013). Mongo 3.2.2 version is being used for this research.

4.2.4 MySQL

MySQL is the most popular open-source relational database management system (RDBMS) (“What is MySQL”, 2016). In July 2013, it was the world's second most widely used RDBMS, and the most widely used open-source client–server model RDBMS (“Oracle and Sun Microsystems”, n.d). It was first released by a Swedish company called MySQL AB in 1995 which was acquired by Sun Microsystems in 2008 and Sun Microsystems too was also acquired in 2010 by Oracle. It is written in C and C++. The MySQL Database Server is very fast, reliable, scalable, and easy to use. MySQL 5.6.17 is the version that is being used for this research.

4.2.5 JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) that allows a Java application to interact with a database. It establishes the connection between the database and the Java application; sends queries to the database; and returns results of the query from the database to the application ("Java SE Technologies – Database", n.d). JDBC has a library called JDBC driver that implements the functions of JDBC and some drivers could modify an application's request so that the request conforms to syntax supported by MySQL (Adeyi, 2014).

The JDBC drivers that are used in this research to connect both databases with the software layer are:

a. Mysql-connector-java-5.1.38 (MySQL)
b. Mongo-java-driver-3.2.2 (MongoDB)

4.3 System Interface

Figure 4.1 is the interface of the system implemented. The interface has a gray background; at the top is text box with a statement above it indicating the user to enter a query in the text box. Below the text box is a blank panel though empty now that shows the mongo query if the query entered is only for the MongoDB. If it involves MySQL, then both the Mongo syntax and its SQL version will appear there.

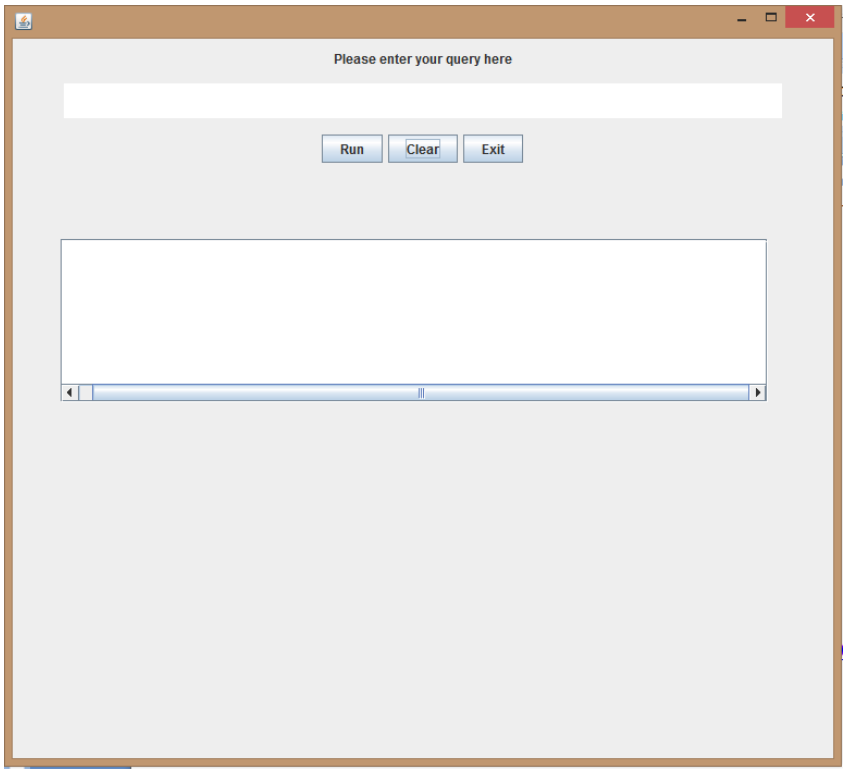


Figure 4. 1: *The Interface for the system*

The big white text box below the panel shows the result of the data set from MongoDB. Should there be a syntax error, it is reported in this same text box. Below the large text box is a blank panel; it displays the data set from MySQL in a tabular form. The subsequent pictures of the interface show the interface with different queries that have been executed.

4.4 Evaluation of System

Rojaikers (2012) adopted SQL as the query language to query both (MySQL and Mongo) databases but in the course of execution and data transformation, some of the data are loss. The same data set are fed into both databases so that the same data sets are expected to be the results for each query that is executed otherwise the software layer. The testing of the system is in two parts. The execution of the CRUD queries which include the insert, read (select/find), update and delete (remove). The next phase of the evaluation is the test of the extended queries.

4.4.1 Evaluation of CRUD queries

Figure 4.2 shows the interface after executing the **db.student.find()*which is requesting for the entire student table(in MySQL) and entire student collection (in MongoDB) to be displayed. The query has an asterisk * prefix, implying it will query both databases.

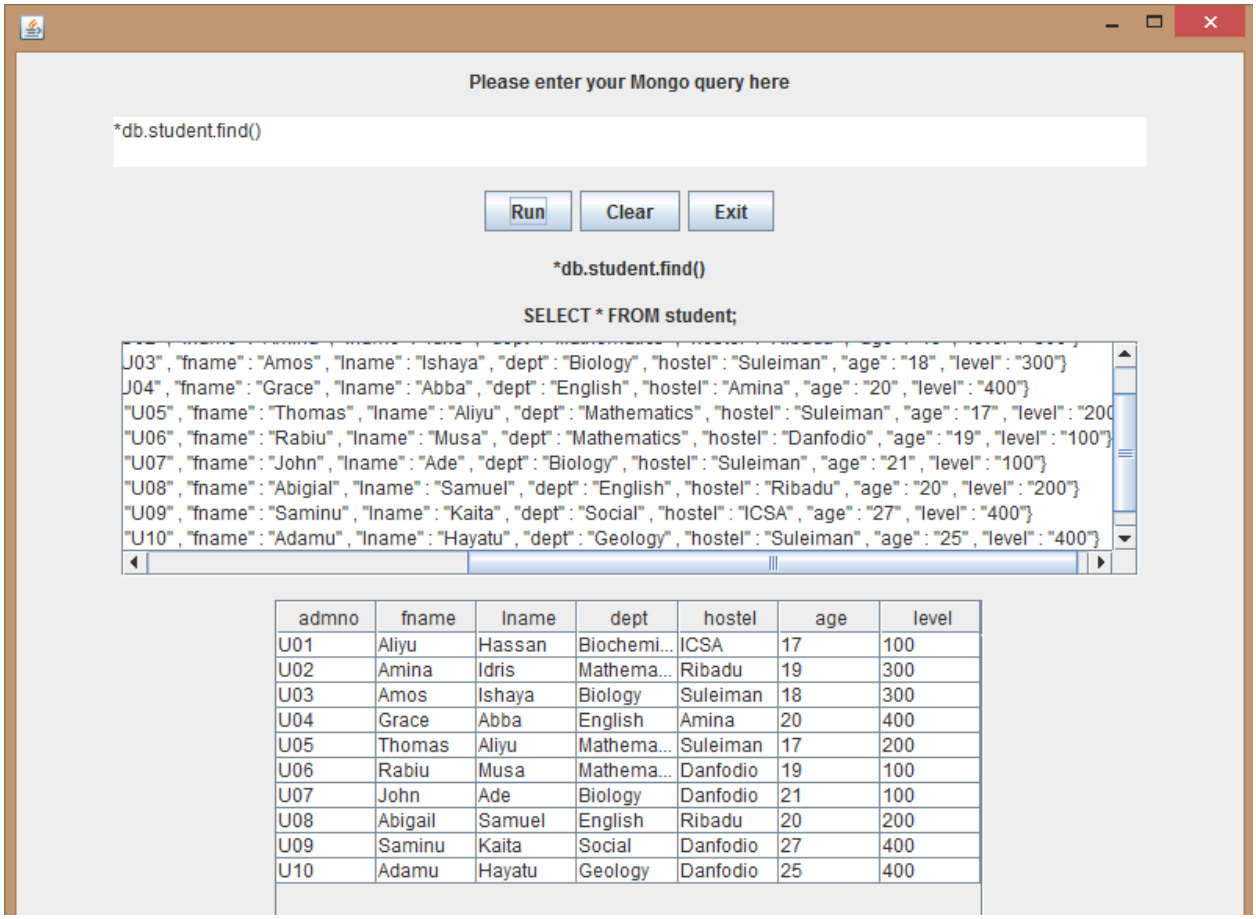


Figure 4. 2: *SELECT ALL query executed on both MongoDB and MySQL*

The result of the query returns the Mongo dataset in the large text box and that of MySQL in the table below the textbox in the interface of figure 4.2. Figure 4.3 is a Mongo only query because it does not have any prefix. The result is the dataset displayed in the large text box only. Figure 4.4 has @ prefix to the mongo syntax which is query for only MySQL; the dataset is returned only the table below the large text box in the interface of figure 4.4.

Further below are the figures of the interface use to carry out INSERT, UPDATE and DELETE operations using the * prefix; which will execute the operations on both databases.

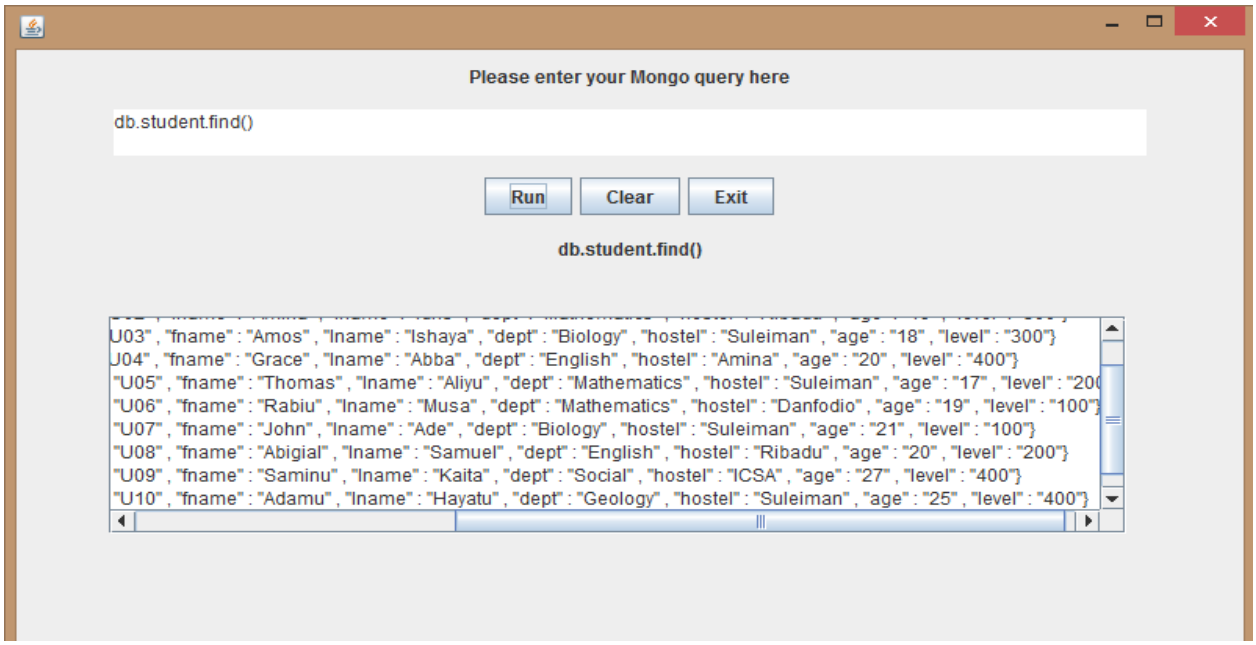
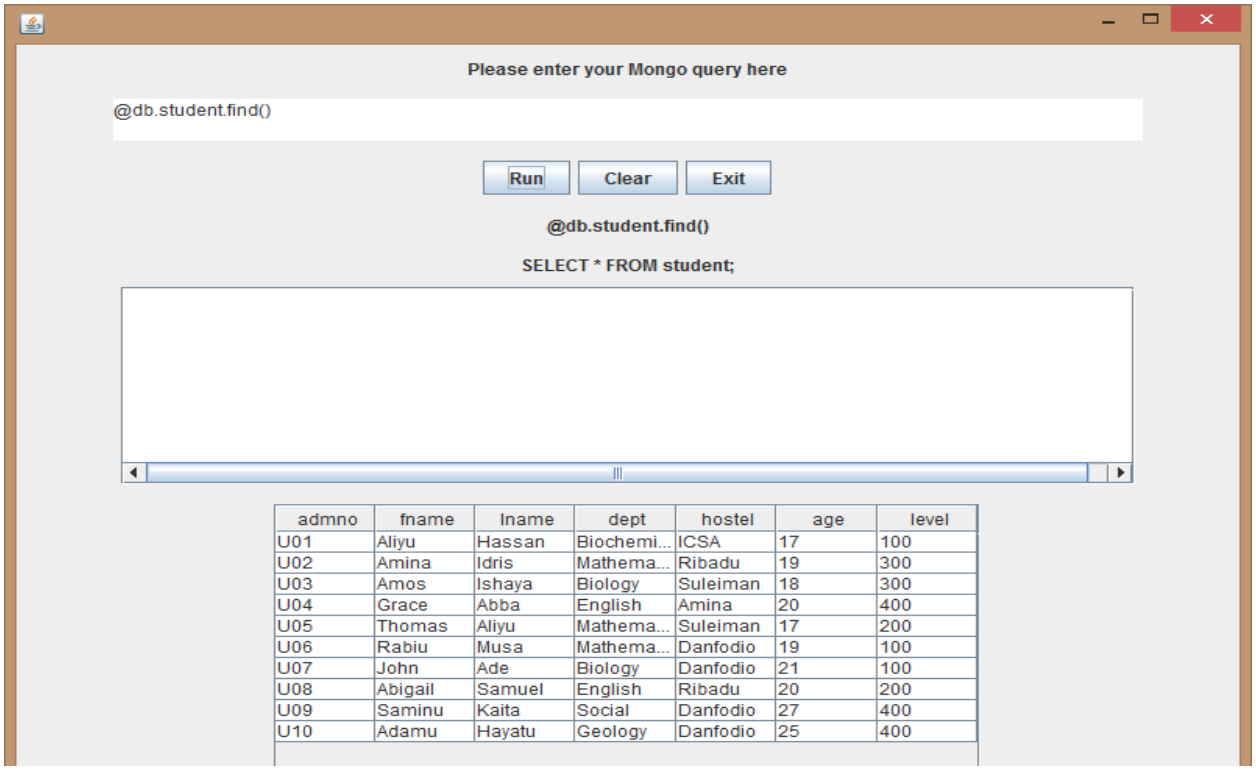


Figure 4. 3: *FIND (Select All) Executed on MongoDB alone*



admno	fname	lname	dept	hostel	age	level
U01	Aliyu	Hassan	Biochemi...	ICSA	17	100
U02	Amina	Idris	Mathema...	Ribadu	19	300
U03	Amos	Ishaya	Biology	Suleiman	18	300
U04	Grace	Abba	English	Amina	20	400
U05	Thomas	Aliyu	Mathema...	Suleiman	17	200
U06	Rabiu	Musa	Mathema...	Danfodio	19	100
U07	John	Ade	Biology	Danfodio	21	100
U08	Abigail	Samuel	English	Ribadu	20	200
U09	Saminu	Kaita	Social	Danfodio	27	400
U10	Adamu	Hayatu	Geology	Danfodio	25	400

Figure 4. 4: *SELECT ALL (Find) Executed on MySQL alone*

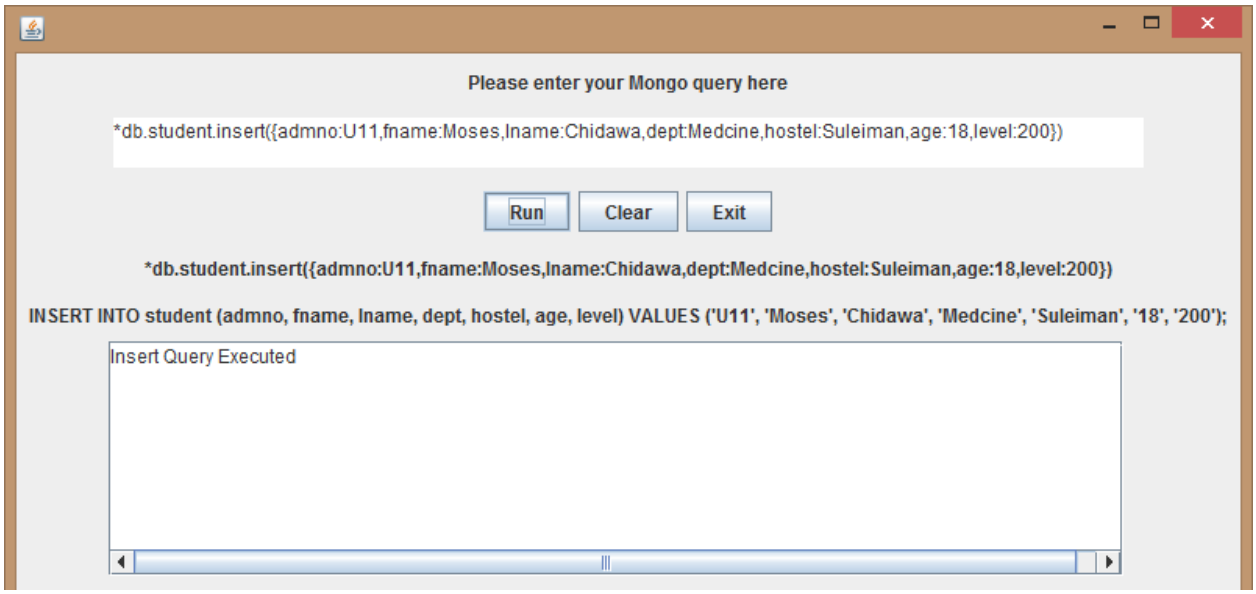


Figure 4. 5: *Insert Query Executed on both Databases (*)*

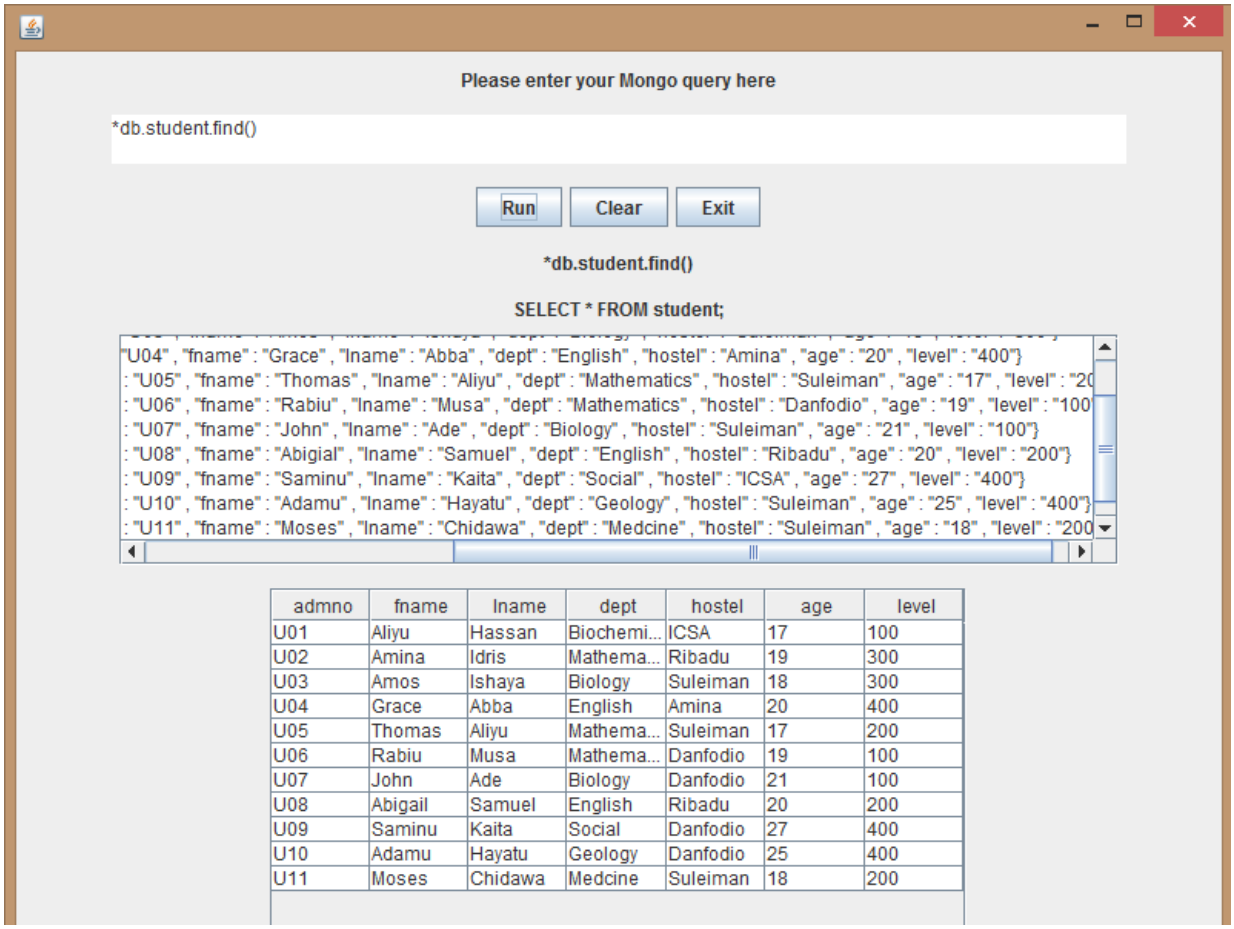


Figure 4. 6: *Confirmation of the Success of the Insert Query (*)*

Figure 4.5 show that an INSERT query has been executed to add one instance of student to both databases. Figure 4.6 confirms that the INSERT query was successful; it shows the new instance with admission number U11 in both the MongoDB and SQL displays on the interface.

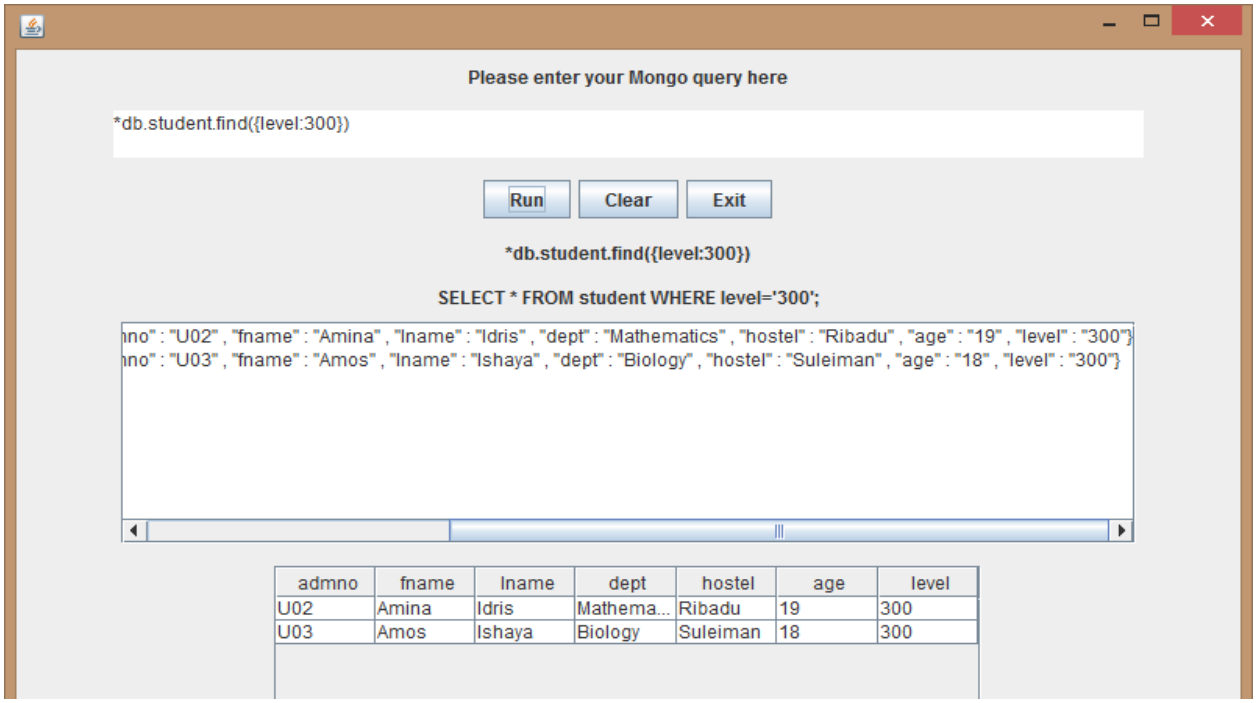


Figure 4. 7: *SELECT (Find) Query with a Criteria Executed on both Databases*

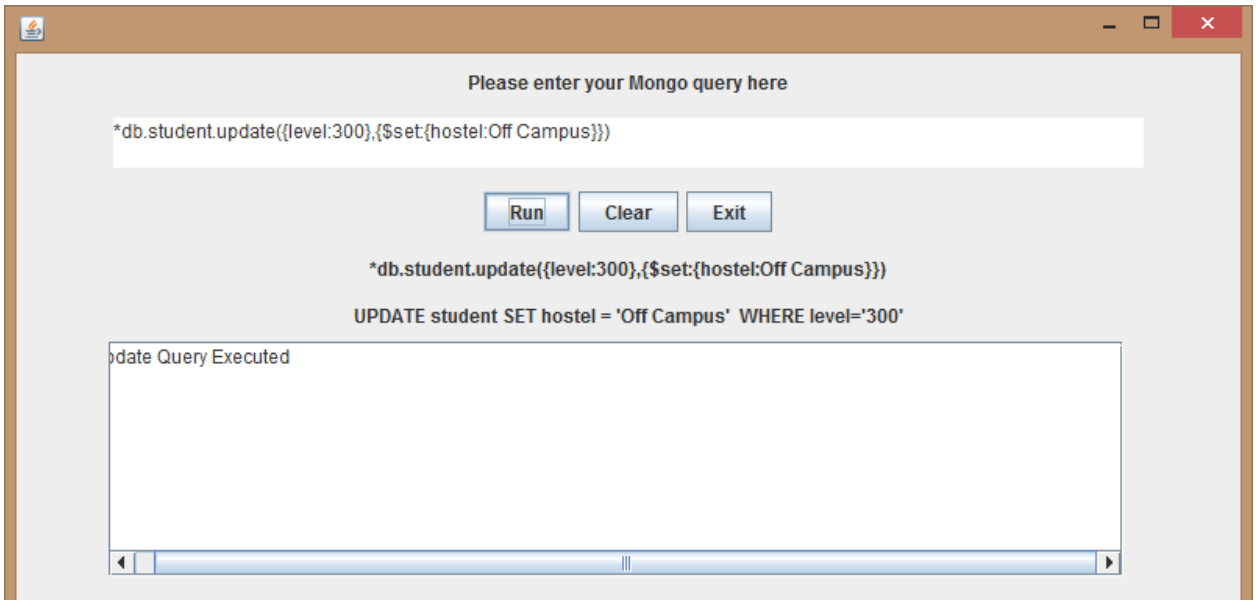


Figure 4. 8: *Update Query Executed on the Instances of the Criteria in Figure 4.7*

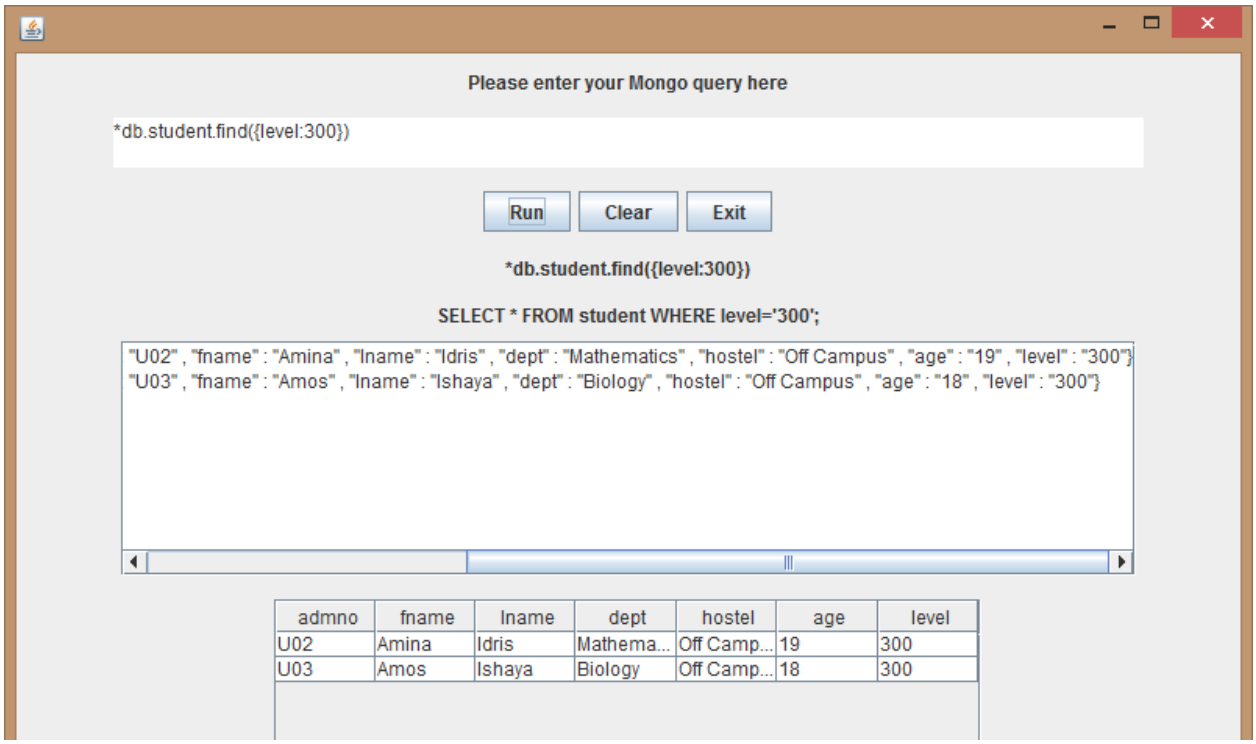


Figure 4. 9: *Confirmation of the Success of the Update Operation in Figure 4.8*

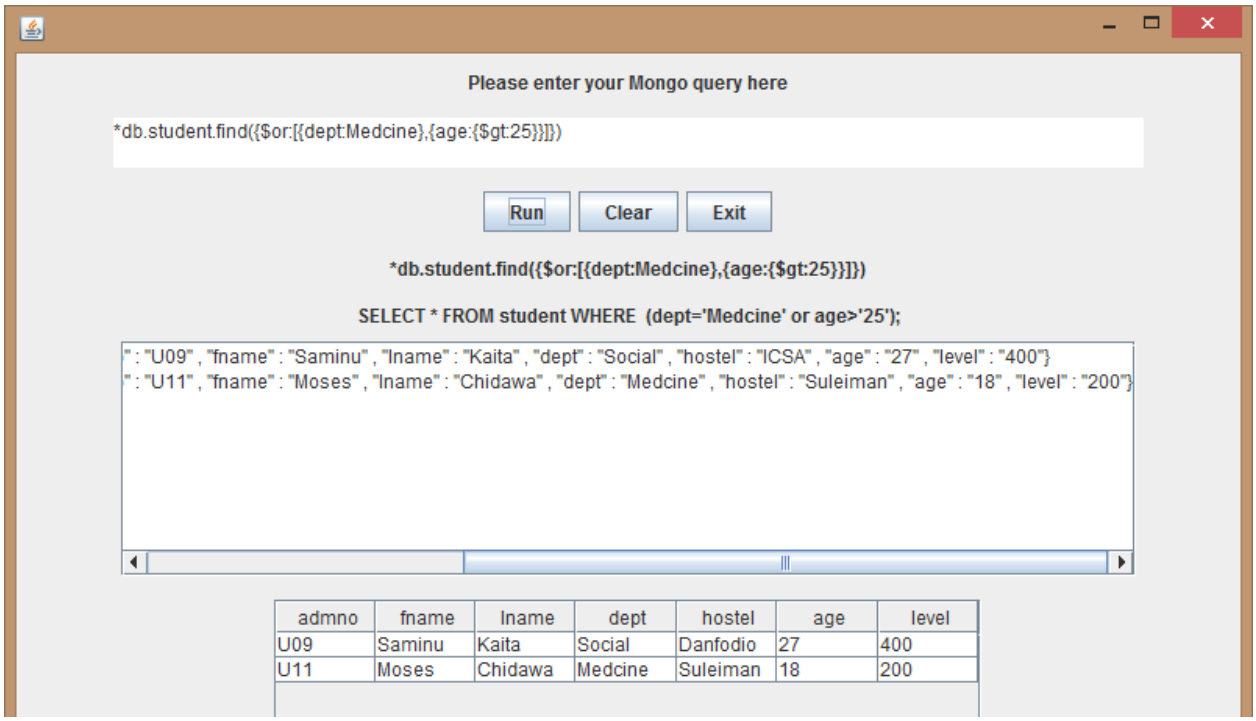


Figure 4. 10: *Another SELECT (Find) query with a Criteria that have been Executed on both Databases*

Figure 4.7 show the instances of student in 300 levels, **db.student.find({level:300})* was executed and an update operation was performed on these instances of data as shown in Figure 4.8 to change the values of the hostel attribute to "off campus". From figure 4.9, it is established that the update operation was successful because the values for the 'hostel' attribute has changed to 'off-campus'.

In figure 4.10, another Select/Find query was executed on both databases to find the instances of students that are either in the department of Medicine or are above 25 years. The instances of student that satisfies the criteria in figure 4.10 are deleted in figure 4.11 because figure 4.12 confirms it; it ran the exact query in figure 4.10 and nothing is displayed because those instances of data have been deleted.

Figure 4.13 is also another confirmation that the update and delete queries were successful from the data that it displays. The query request for the entire student table (SQL) and collection (MongoDB). The result displayed shows the 300 level students have values for hostel to be 'off-campus' as earlier edited. The instances of U09 and U11 are not in the result because they have been deleted.

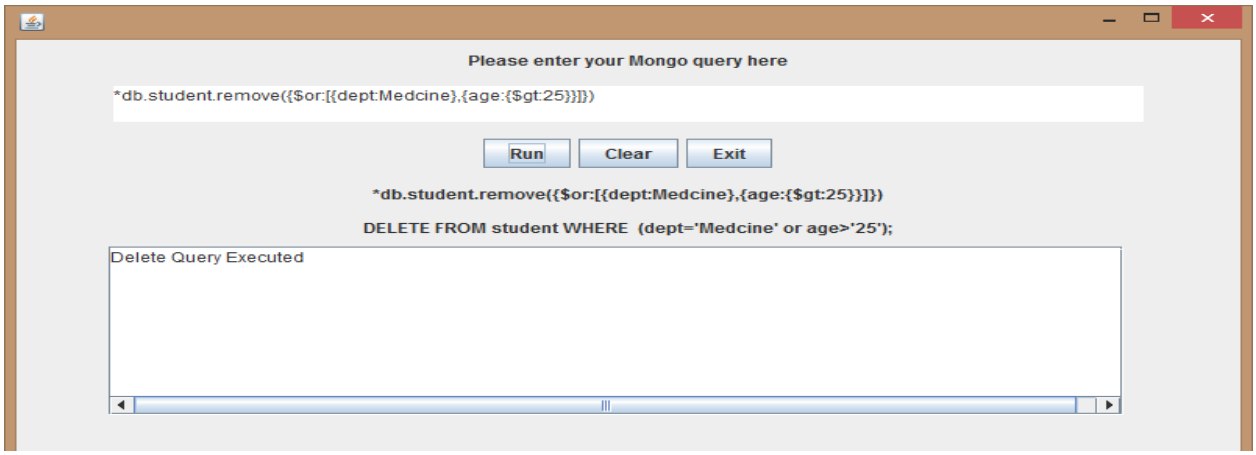


Figure 4. 11: *Delete (Remove) Query executed on the Instances of the Criteria in Figure 4.10*

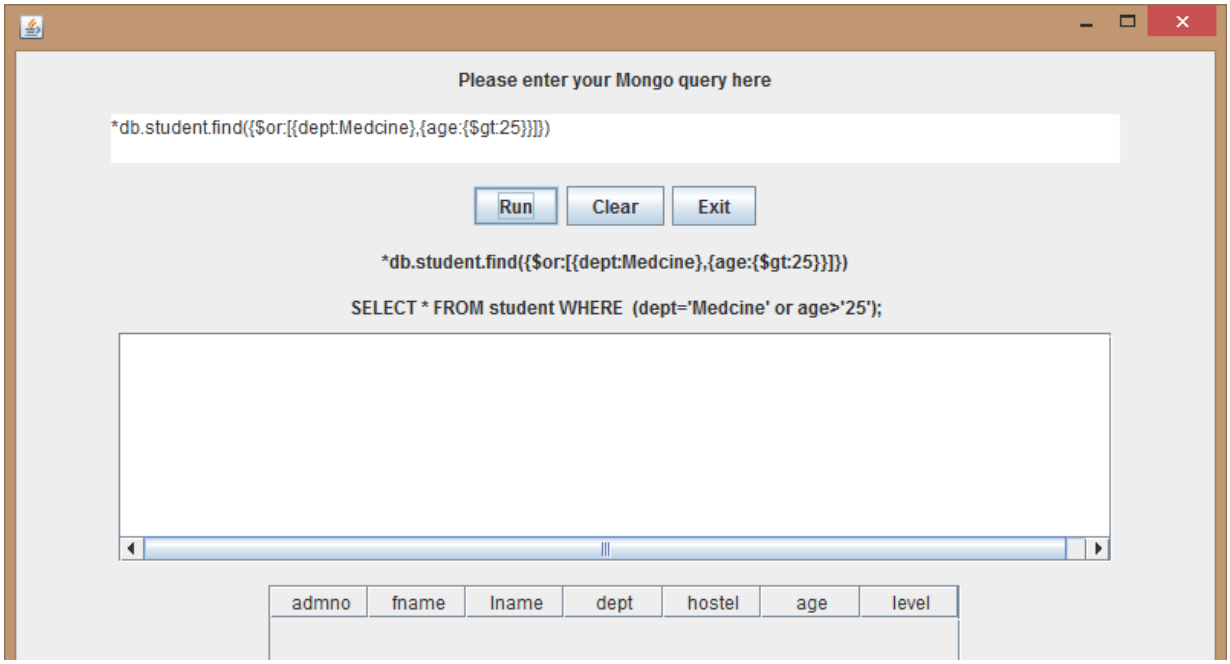


Figure 4. 12: *Confirmation of the Success of the Delete Query in Figure 4.11*

4.4.2 Evaluation of Extended Queries

The extended Mongo queries to test are: create table, sub-query and the join query. These queries have the @ prefix because they are operations for MySQL to execute. Beginning with the CREATE TABLE query use to create a table in the relational database before data insertion can commence. Figure 4.14 shows the create table that has been executed to create a table called "courses" that is made up of four (4) columns: title, code, cu (credit unit), dept (department) and the code is the adopted as the Primary Key. Figure 4.15 ran *@db.courses.find()* to show the entire 'courses' table, but no data could display but the structure of the table because, the 'courses' table has just being created with no data inserted yet.

The next extended query is the sub query. Figure 4.16 shows a sub query that has been executed. The query is asking to see the students from the student table who has registered the course MATH105; the verification of the course registration is carried out in another table called 'coreg' (coreg is the table for storing **co**urse **reg**istration). This sub query uses the *admno* attribute in the *student* table to verify which students registered MATH105 in *coreg* table. Figure 4.17 shows the coreg table and the registration of the MATH105 course. This sub query in figure 4.16 has used projection (the second curly brackets) to specify three attributes to be displayed; they are *admno, fname* and *lname.*

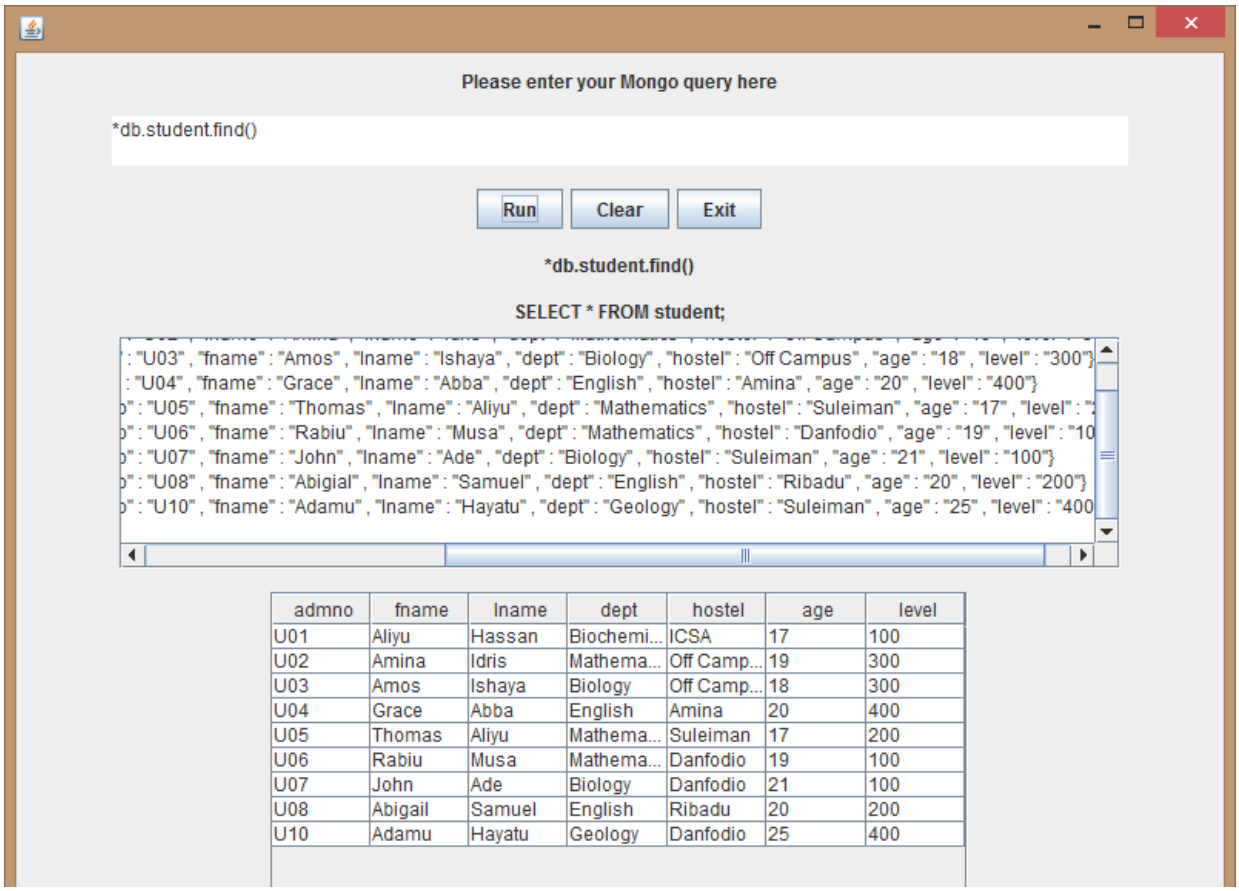


Figure 4. 13: *Another Confirmation for the success of the Update (Figure 4.8) and Delete (Figure 4.11) Queries*

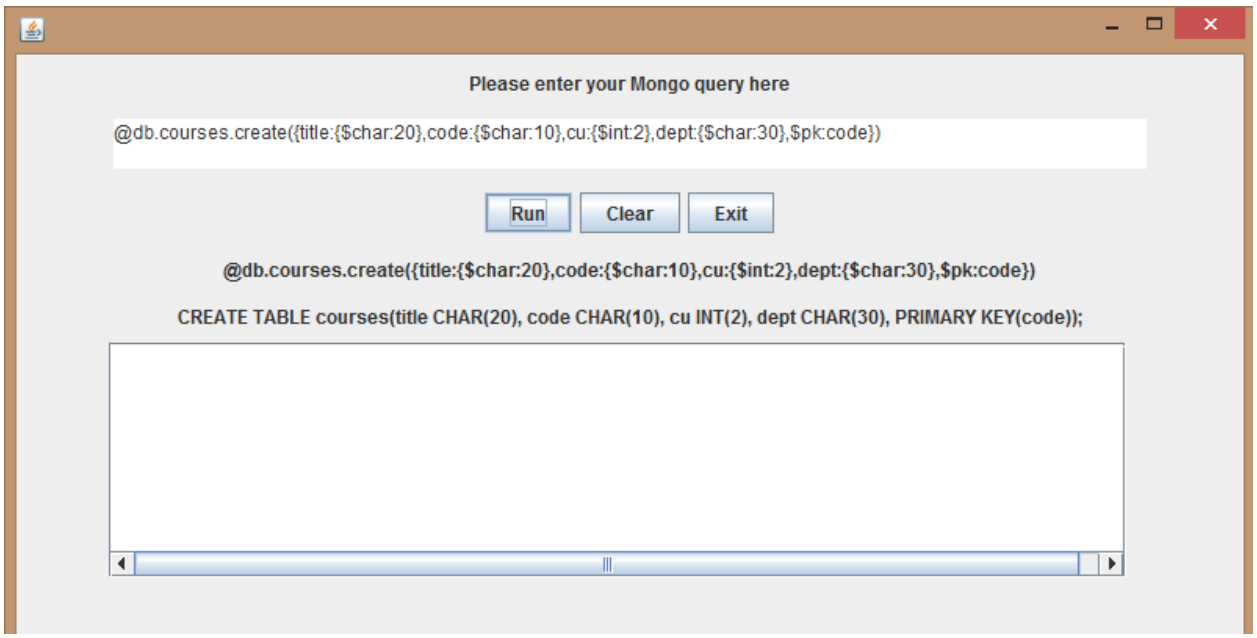


Figure 4. 14: *CREATE TABLE query executed on MySQL (@)*

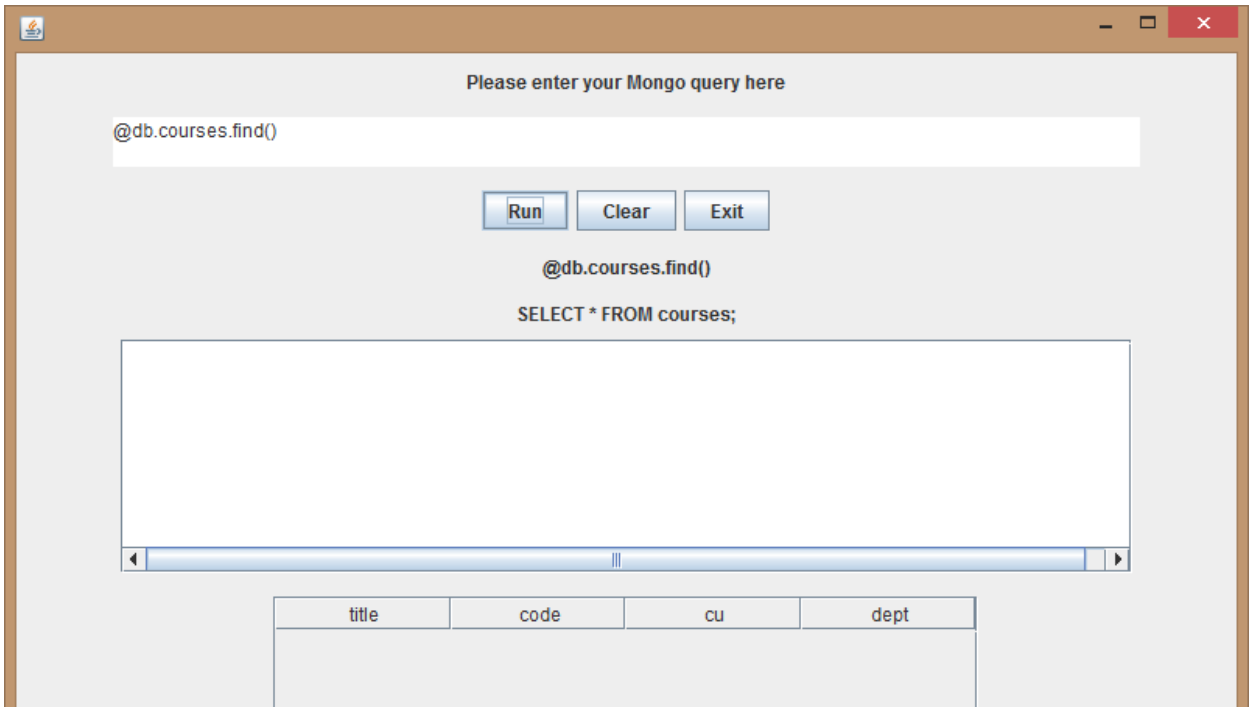


Figure 4. 15: *Confirmation for the Success of the CREATE TABLE Query*

The Join query allows data to be fetched from more than one table in the relational database. The query used in figure 4.18 seeks data from three tables (student, coreg and course) with matching values of specified attributes across the tables; where *admno* in student table matches *admno* in coreg table and where *code* in coreg table matches *code* in course table. In SQL's Join query, since it involves more than a table, it is expected that every attribute in the join query statement must be appended to the table name using a period (.). For example, *admno* in the *student* table should appear as 'student.admno'. In projection curly bracket of join query in figure 4.18, the attribute names are required to be appended with their respective tables that is why in the Mongo syntax of figure 4.18 uses the plus sign (+) to append attribute name to the table name. The join query is long that the whole of it cannot be captured figure 4.18. This is the complete query: @db.student+coreg+course.find({$join:[{student:admno,coreg:admno},{coreg:code,course:code }]},{student+lname:1,student+fname:1,coreg+code:1,course+title:1,course+semester:1})

And the SQL equivalent is:

SELECT student.lname, student.fname, coreg.code, course.title FROM student JOIN coreg JOIN course WHERE student.admno=coreg.admno AND coreg.code=course.code

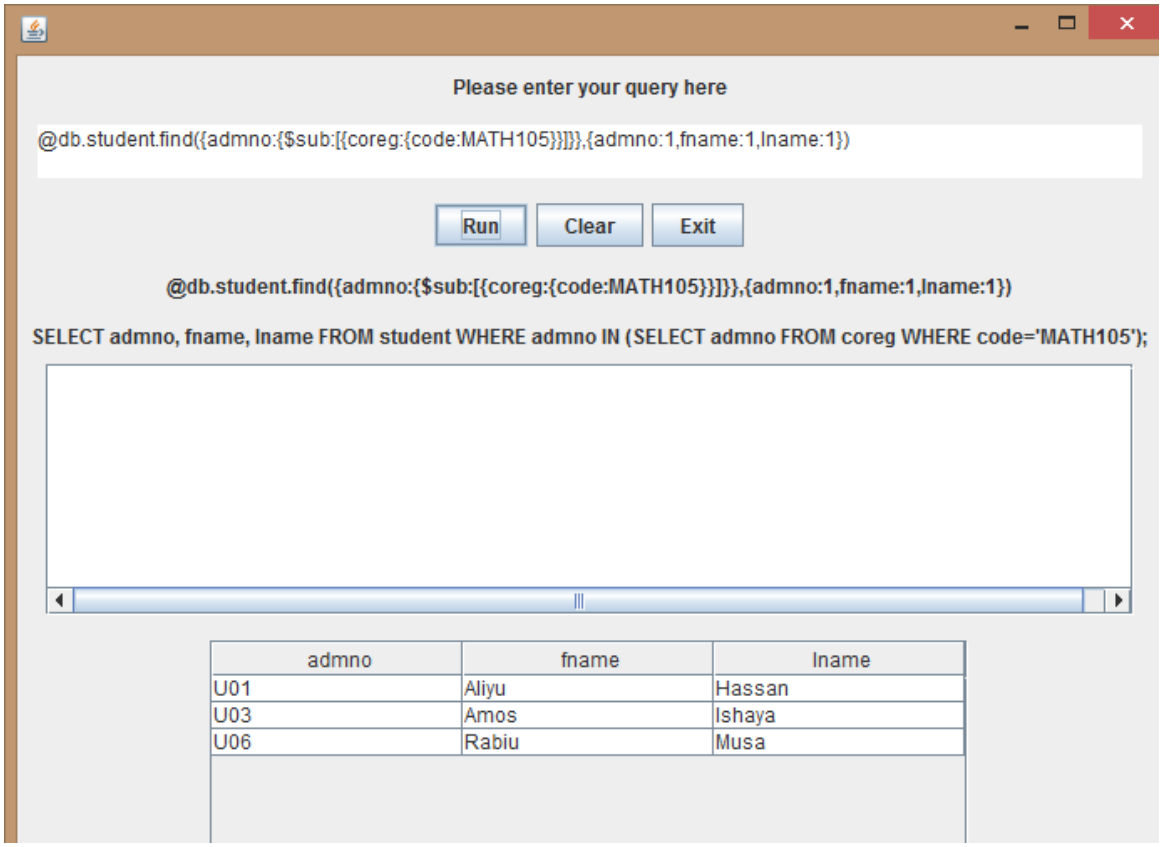


Figure 4.16: *Sub-query executed on MySQL*

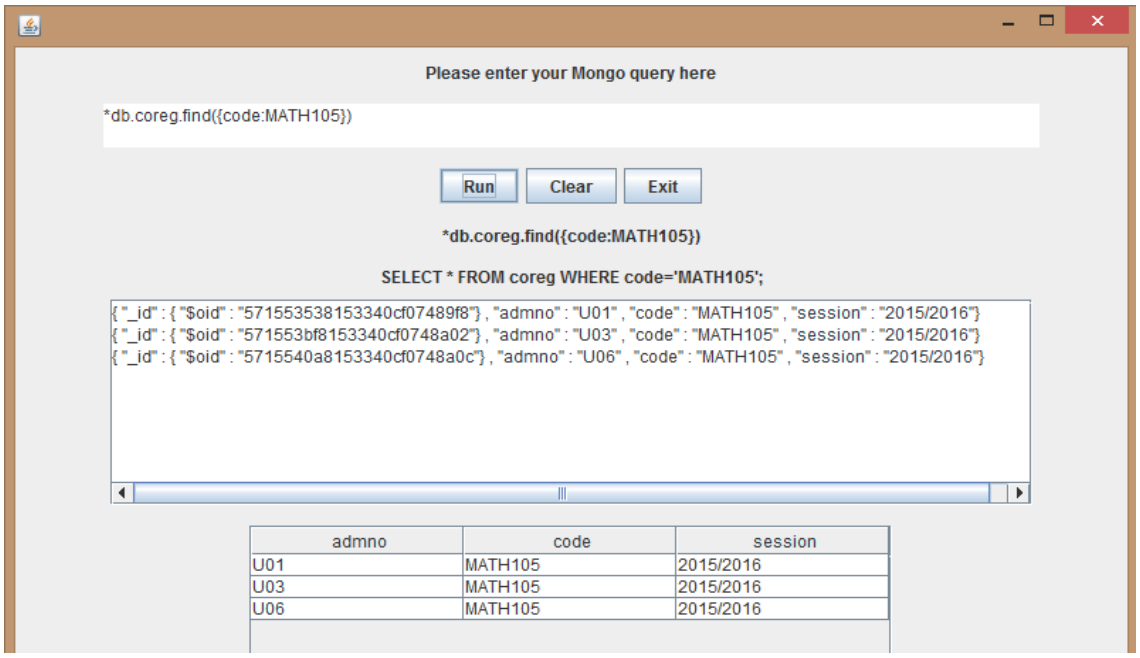
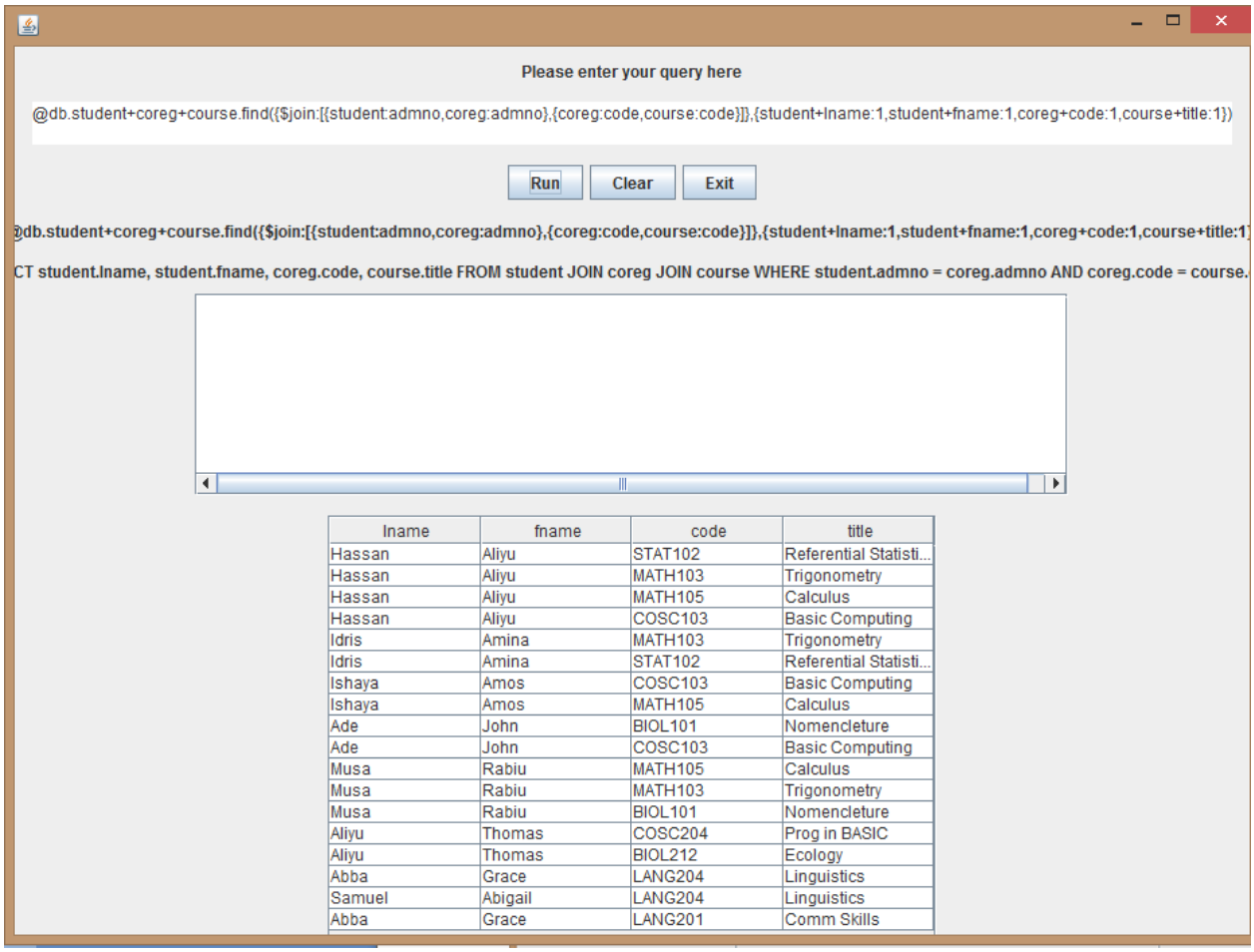


Figure 4. 17: *coreg table Showing the Registration of MATH105 course*



lname	fname	code	title
Hassan	Aliyu	STAT102	Referential Statisti...
Hassan	Aliyu	MATH103	Trigonometry
Hassan	Aliyu	MATH105	Calculus
Hassan	Aliyu	COSC103	Basic Computing
Idris	Amina	MATH103	Trigonometry
Idris	Amina	STAT102	Referential Statisti...
Ishaya	Amos	COSC103	Basic Computing
Ishaya	Amos	MATH105	Calculus
Ade	John	BIOL101	Nomencleture
Ade	John	COSC103	Basic Computing
Musa	Rabiu	MATH105	Calculus
Musa	Rabiu	MATH103	Trigonometry
Musa	Rabiu	BIOL101	Nomencleture
Aliyu	Thomas	COSC204	Prog in BASIC
Aliyu	Thomas	BIOL212	Ecology
Abba	Grace	LANG204	Linguistics
Samuel	Abigail	LANG204	Linguistics
Abba	Grace	LANG201	Comm Skills

Figure 4.18: JOIN query executed on three relational (MySQL) tables

4.5 Summary

The tools that were employed to implement the system for this research were listed and briefly explained. Since the same data is stored in both databases, the figures showing the executed queries confirms that query from Mongo syntax to SQL was correctly converted and executed by showing the same datasets in both the Mongo display and SQL display on the interface.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary and Conclusion

This research is aimed at providing a software layer above the two different databases (SQL and NoSQL) so that only one query language can be used to query both databases. This reduces the work of the users by having to learn only one language instead of learning more than one language to query the different databases in a hybrid database.

The software layer (system) was designed and implemented. The design includes the conversion of the MongoDB syntax to SQL syntax and the extension of Mongo syntax to cater for SQL queries that do not have equivalents in MongoDB due to their difference in data model and technology.

The system was implemented and tested. The results obtained from testing showed the software layer has achieved the aim based on the scope of the query sets defined (CRUD) in section 3.5 of this research. The same dataset was used on both databases to evaluate the system. The result gotten from the evaluation proves that the implementation was successful.

The contribution to knowledge from this research is: it is possible to make work easier for a hybrid database administrator by providing a platform through which the databases that make up the hybrid database can be queried using a one query language without losing data from any of the databases.

5.2 Recommendations

This research has a scope defined. It achieved its aim based on only CRUD operations which are some basic operations. It can be recommended in the future that further researches to extend this work should include:

a. Work to cover more Mongo command other than the CRUD commands that defined the query scope for this research.
b. Work that has more than one NoSQL database in the hybrid system. The hybrid database could have the relational database and one database from each of the four types of databases (Key/Value, Document, Column family and Graph databases).
c. Work that can accept commands either in SQL or NoSQL syntax and the software layer translates the commands to the query languages of the respective databases in the hybrid system.

REFERENCES

Abramova, V., Bernardino, O., and Furtado, P. (2014). Which NoSQL database? A Performance Overview. *Open Journal of Databases (OJDB)* 1(2), pp. 17-24.

Adeyi, T. S. (2014). Design and implementation of DualFetchQL for SQL and NoSQL databases. (Unpublished *Master thesis).* Ahmadu Bello University, Nigeria.

Artem, C., Shyong L., and Farshad F. (2009). Semantics preserving SPARQL-to SQL translation. *Data & Knowledge Engineering*. 68(10): 973–1000.

Cattell, R. *(2010).*Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record,* 39(4): 12-27

Comparision Query Operators (n.d). Retrieved from: https://docs.mongodb.com/manual/reference/operator/query-comparison/

Couchbase –Why NoSQL. (n.d.). Retrieved 2016 April 7, from http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

DB engine comparison: The top 10 SQL and NoSQL databases (2014). Retrieved from https://jaxenter.com/the-top-10-sql-and-nosql-databases-108072.html

DB-Engines Ranking of Document Stores. (Jan. 2016). Retrieved from: http://dbengines.com/en/ranking/

Ferreira, L. (2012). Bridging the Gap Between SQL and NoSQL: SQL and ACID over a VLSD. *Master's thesis, Universidade do Minho,* Portugal. pp. 4 - 18.

Harris, D. (2013). 10gen Embraces what it created, becomes MongoDB Inc. Retrieved from: https://gigaom.com/2013/08/27/10gen-embraces-what-it-created-becomes-mongodb-inc

ITJobsWatch NoSQL, (2016). Retrived 2016 January 12 from: http://www.itjobswatch.co.uk/jobs/uk/nosql.do

Jafarpour, H., Tatemura, J., and Hacigumus, H. (2015, March). Transactional Replication in Hybrid Data Store Architectures. *18th International Conference on Extending Database Technology (EDBT 2015), Brussels, Belgium,* pp. 569-580

JAXenter Editiorial Team (2014). DB engine comparison: The top 10 SQL and NoSQL databases. Retrieved 2016 January 10, from: https://jaxenter.com/the-top-10-sql-and-nosql-databases-108072.html

Nance, C., Travis, L., Iype, R., and Harmon, G. (2013, March). NoSQL vs. RDBMS - Why There is Room for Both. *Proceedings of the Southern Association for Information Systems Conference, Savannah, GA, USA*. 1: 5 – 7.

Nayak, A., Poriya, A., and Poojary, D. *(2013).* Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems, New York, USA.* 5(4): 16-19.

Oracle and Open Source. Retrieved 2016, April 15, from http://www.oracle.com/us/024981.pdf

Oracle and Sun Microsystems (n.d.). Retrieve from: https://www.oracle.com/sun/index.html

Oracle. (n.d). Java Se Technologies. Retrieved 2016 April 10, from: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

Polo, M., Piattini, M., and Ruiz, F. (2001): Reflective persistence (Reflective CRUD: reflective create, read, update and delete). Sixth European Conference on Pattern Languages of Programs (EuroPLOP). Retrieved from: http://hillside.net/europlop/HillsideEurope/Papers/EuroPLoP2001/2001_PoloEtAl_ReflectivePersistence.pdf

Quora (2013). What's the most popular NoSQL Database. Retrieved 2016 January 8, from: https://www.quora.com/Whats-the-most-popular-NoSQL-database

Roijackers, J. (2012). Bridging SQL and NoSQL. *Master's thesis, Eindhoven University of Technology*, Netherlands. pp. 1 – 10, 15 – 60.

Ron, A., Shulman-Peleg, A., and Bronshtein, E. (2015), No SQL, No Injection? Examining NoSQL Security, IBM, Beer Sheba, Israel

Sadalage, P. J., and Fowler, M. (2012) NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Upper Saddle River, New Jersey: Addison-Wesley. Retrieved from http://resource.mitfiles.com/CSE/IV%20year/VII%20sem/Program%20Elective/Big%20Data%20Analytics/books%20and%20slides/books/NoSQL%20Distilled.pdf

Sadalage, P., (2014). Retrieved 2016 April 10, from: https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Sian, R., (n.d.) Mapping SQL to MongoDB. Retrieved 2016, January 8 from: s3.amazonaws.com/info-mongodb-com/sql_to_mongo.pdf

SQL - RDBMS Concepts. (n.d.). Retrieve 2016 April 10, from: http://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm

SQL Transactions. (n.d.). Retrieve 2016 April 11, from: http://www.tutorialspoint.com/sql/sql-transactions.htm

Varley, I.T. (2009).No Relation: The Mixed Blessings of Non-Relational Databases. M.A thesis. The University of Texas, USA. pp. 2, 4-12.

Welcome to NetBeans community. (2016). Retrieve from: https://netbeans.org/welcome.html