11-2017

Migrating From SQL to NoSQL Database: Practices and Analysis

Fatima Jamal Al Shekh Yassin

UAEU

جامعة الإمارات العربيـة المتحدة
United Arab Emirates University

United Arab Emirates University

College of Information Technology

Department of Computer Science and Software Engineering

MIGRATING FROM SQL TO NOSQL DATABASE: PRACTICES AND ANALYSIS

Fatima Jamal Al Shekh Yassin

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering

Under the Supervision of Dr. Mamoun Adel Awad

November 2017

Declaration of Original Work

I, Fatima Jamal Al Shekh Yassin, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled "*Migrating from SQL to NoSQL Databases: Practices and Analysis*", hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Mamoun Awad, in the College of Information Technology at UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature: ____________________ Date: 28. 12. 2017

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Dr. Mamoun Adel Awad

   Title: Associate Professor

   Department of Computer Science and Software Engineering

   College of Information Technology

   Signature ____________________ Date 16/11/2017

2) Member: Salah Bouktif

   Title: Associate Professor

   Department of Computer Science and Software Engineering

   College of Information Technology

   Signature ____________________ Date 16/11/2017

4) Member (External Examiner): Zaher Al Aghbari

   Title: Professor / Chair of The Department of Computer Science

   Department of Computer Science

   Institution: University of Sharjah , UAE

   Signature ____________________ Date 16/11/2017

This Master Thesis is accepted by:

Dean of the College of Information Technology: Professor Khalid Shuaib

Signature __________ Date 19/12/2017

for Dean of the College of Graduate Studies: Professor Nagi T. Wakim

Signature Ali Hassan Date 28/12/2017

Copy 2 of 7

Declaration of Original Work

I, Fatima Jamal Al Shekh Yassin, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled "*Migrating from SQL to NoSQL Databases: Practices and Analysis*", hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Mamoun Awad, in the College of Information Technology at UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature: __________________________ Date: ________________

Abstract

Most of the enterprises that are dealing with big data are moving towards using NoSQL data structures to represent data. Converting existing SQL structures to NoSQL structure is a very important task where we should guarantee both better performance and accurate data. The main objective of this thesis is to highlight the most suitable NoSQL structure to migrate from relational Database in terms of high performance in reading data. Different combinations of NoSQL structures have been tested and compared with SQL structure to be able to conclude the best design to use. For SQL structure, we used the MySQL data that is stored in five tables with different types of relationships among them. For NoSQL, we implemented three different MongoDB structures. We considered combinations of different levels of embedding documents and reference relationships between documents. Our experiments showed that using a mix of one level embedded document with a reference relationship with another document is the best structure to choose. We have used a database that contains five tables with a variety of relationships many-to-one, and many-to-many. Also the huge amount of data stored in all the structures about 2 millions record/document. The research compares clearly between the performance of retrieving data from different MongDB representation of data and the result shows that in some cases using more than one collection to represent huge data with complex relationships is better than keeping all the data in one document.

Keywords: Big data, SQL, NoSQL, MySQL, MongoDB, Embedding document, Reference relationship, one-to-one, many-to-one, many-to-many.

Title and Abstract (in Arabic)

التحويل من قاعدة البيانات سكل الي نو سكل : تطبيقات و تحيليل

الملخص

معظم الشركات التي تتعامل مع البيانات الكبيرة تتحرك نحو استخدام هياكل البيانات الجديدة نوسكل (NoSQL) لتمثيل بياناتها. تعتبر عملية تحويل هياكل البيانات الحالية سكل (SQL) الي البنيه الجديدة نوسكل (NoSQL) عملية مهمة جدا حيث يجب ضمان الحصول على أداء افضل و بيانات دقيقة بعد تحويل البيانات الى البنيه الجديدة. الهدف الرئيسي من هذه الرسالة هو تسليط الضوء على انسب تصميم للبيانات باستخدام قاعدة البيانات مونغو (Mongo) ومقارنة ادائها مع تصاميم اخرى لنفس البيانات وكذلك مقارنتها بأداء قاعدة البيانات نوسكل (MySQL). لقد قمنا بتمثيل البيانات بداية باستخدام ماي سكل (MySQL) في خمس جداول بينها علاقات مختلفة من حيث النوع والكم، ثم استخدمنا خمسة انواع مختلفة التعقيد من الاوامر لاستخراج المعلومات من هذه الجداول. وكذلك قمنا باستخدام ثلاثة تصاميم مختلفة لتمثيل البيانات ذاتها باستخدام قاعدة البيانات مونغو (Mongo)، وايضا استخدمنا نفس الاوامر لاستخراج المعلومات نفسها من التصاميم الثلاثة. بعد الانتهاء من تسيجل النتائج قمنا بمقارنتها لنتوصل في نهاية البحث الى ان استخدام تصميم يحتوي على نوعين من المستندات هما مستوى رئيسي من البيانات يحتوي مستوى فرعي لبيانات مرتبطة بالمستوى الرئيسي وهذه العلاقة تعرف بالبيانات المضمنة و مستند ثان تربطه علاقة مرجعية مع المستند الاول وهو يحتوي على تتمت البيانات المطلوبة. لقد استخدمنا في هذا البحث خمسة جداول بينها جميع انواع العلاقات (واحد لكثير one ot many، كثير لكثيرmany to many) كما ان عدد العلاقات بين الجداول يتراوح بين علاقه واحده او علاقتين او ثلاث علاقات. كما تم ايضا تخزين كم هائل من البيانات في جميع التصاميم المطروحة حوال 2 مليون.

مفاهيم البحث الرئيسية: البيانات الكبيرة، سكل (SQL)، نوسكل (NoSQL)، ماي سكل (MySQL)، مونغو (Mongo)، البيانات المضمنه، علاقة مرجعية، واحد لواحد (one to one)، واحد لكثير (one ot many)، كثير لكثير(many to many).

Acknowledgements

I would first like to thank my thesis advisor Dr. Mamoun Awad. The door to Dr. Awad office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to acknowledge Mrs. Hanady Ghazal as she was the english proof reader of this thesis, and I am gratefully indebted to her for her very valuable comments and support.

Finally, I must express my very profound gratitude to my husband Belal Hamdan, my kids Hasan and Nour and to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Dedication

To my beloved parents and family

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Program Interface
BASE	Basically Available, Soft state, Eventual consistency
BSON	Binary JSON
CPA	Consistency, Partition-tolerance and Availability
DB	Database
DBMS	Data Base Management System
DDR3	Double Data Rate Type 3
ERD	Entity Relationship Diagram
GB	Giga Bytes
GHz	Gigahertz
GPL	General Public License
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
MHz	Megahertz
NoSQL	Not only SQL
OODB	Object Oriented Database
OODBMS	Object Oriented Database Management System
OOP	Object Oriented Programming
OS	Operating System
REST	Representational State Transfer
SQL	Structured Query Language

Chapter 1: Introduction

1.1 Overview

As the world trends are moving towards having applications with cloud computing, advancement of IT industry, web applications, internet of things and big data [18, 21], enterprises are mostly using NoSQL instead of relational DB. The adoption of NoSQL DB is the response of the growth of data that requires faster data access and analysis [20]. For example big data, which is very huge and unstructured, requires powerful machine to process. Also it needs distributed systems to contain it, and flexible data schemas to design it. The NoSQL DB appeared to fit the needs of the new market and to satisfy the limitation of the relational DB [15].

The use of NoSQL DB in new enterprises is not a major issue because the new application design will be based on NoSQL DB. But the problem appears when the existing systems that relay on relational DB are restructuring their systems to implement NoSQL DB. They need to reanalyze the system requirements to build up the new DB schema [4].

The migration of the legacy system to a new system and maintaining the same functionality and data integrity of the legacy system is an important challenge for enterprises. The migration process has two requirements. First, changing the design and second, data migration [1].

The new design of the DB can be achieved either by an expert of the new paradigm that will redesign the existing DB to the new system or by using tools to automatically convert the old schema to a new one. Many researches have proven that the relational DB can be converted into NoSQL DB. As well as many tools were developed in a variety of research projects to achieve data migration by looking at the problem from a different point of view [1, 2]. The noticeable thing in most of the researches that presented DB schema design migration tools is that they have tested their tools on one database only. Most of them did not show enough results that guarantee keeping the same performance and reliability of the data. Additionally, they used simple database implementation that contains maximum of 3 tables and with few relationships between tables. Furthermore, the database was filled in with few records that did not exceed a million records [1, 2, 11].

The data migration tool is simpler and easier to achieve than the schema conversion tool. Therefore data can be imported easily to a certain format and exported simply to the new design.

The main purpose of the research is to prove that the performance of the new NoSQL system in reading data is better than the relational DB. In this research, migrating the relational data schema into 3 different NoSQL schemas are designed manually and testing the performance of reading data in the new systems after moving the same data into the new systems was the main focus. The data reading performance of the new systems was tested, checked and compared to the old system. The same relational DB schema used in [17] was used in this research, but different NoSQL database representations for the new systems. One collection of MongoDB was used to represent my relational DB with two levels embedding documents, another one with reference relationship and one level embedding documents, and the last one with reference relationships between five collections. Different types of queries with different complexity on all the designs was executed.

The queries vary from very simple queries to complicated queries that involve different levels of joins and aggregation function use.

The contribution of this research includes, converting a huge amount of data that is stored in 5 tables in MySQL DB to the best design in MongoDB. The selected DB contains 5 tables with different numbers and types of relationships between them. We redesigned the relational DB schema to 3 different MongoDB schemas considering the level of embedding documents and reference document and the number of collections generated.

The research assumes using MySQL server version 5.7.17 MySQL Community Server (GPL) for relational database and MongoDB version v3.4.1. Both DBs are running on MacBook Pro (13-inch, Mid 2010) with 2.4 GHz Intel Core 2 Duo processor, 4 GB 1067 MHz DDR3, NVIDIA GeForce 320M 256 MB, and the version of OS installed in the devise is macOS Sierra Version 10.12. Java is used to develop the API for managing the communication with the DBs through NetBeans IDE 8.2. The library used to access MySQL server DB for java API is “mysql-connector-java-5.1.40-bin.jar”, and the other one used to access MongoDB is “mongo-java-driver-3.4.1.jar”.

The research is evaluated by developing Java API that filled the tables in MySQL server database with random data and then by executing five different level queries on the data. The time consumed to retrieve all the data was recorded for each query. Another three API were developed to fill the MongoDB documents in the three different schemas, and the same five queries were tested to calculate the time needed to complete the execution of the queries on the MongoDB.

1.2 Statement of the Problem

The huge amount of streaming data available nowadays are due to massive use of mobile computing, cloud computing, IoT, and other new technologies. Such tremendous amounts of data add a great deal of challenges to the traditional relational DB paradigm. Those challenges are related to performance, scalability, and distribution. To over come those challenges enterprises start to move towards implementing new DB paradigm known as NoSQL.

So given a well-designed relational DB, *S*, we would like to transform *s* into *s'*, where *s'* is the NoSQL structure that achieve the best performance among all other structures, *S*. In other words, *argmax F(s)*, where $s \in S$, S is the set of all possible NoSQL structures and *F* is a function to maximize. In this thesis, *S* has 3 different structures, and *F* is a function of retrieval time of data queries, i.e., manipulation queries are not considered.

When an enterprise makes the decision to move to the new NoSQL DB, it should make sure that the migration will improve the performance of the system in addition to speed up data processing.

1.2.1 Problem definition

After the decision of converting a current relational DB in an enterprise into NoSQL DB is made, the argument about which type of NoSQL DB should be used will start. In NoSQL DB, data can be presented in different formats. Some of the NoSQL DBs use document representation of data; other types use column representation, key-values are used in some types of NoSQL DB and some use graphs to represent data. After the enterprise chooses the most suitable NoSQL DB type that meets their needs, the best design that will improve the performance of the system must be selected to start the migration process later.

In this research, MySQL server DB that contains data related to employees stored in 5 tables with different types of relationship between them. The data was represented using NoSQL DB Mongo that stores data as documents. The data reading performance of 3 different designs of the Mongo were compared with each other and with the MySQL DB. The 3 designed were selected to cover the different relationships between documents in MongoDB which are fully embedded documents with different levels of embedding, reference relationship between two documents that each of them has one level of embedding document, and completely reference relationships between 5 documents.

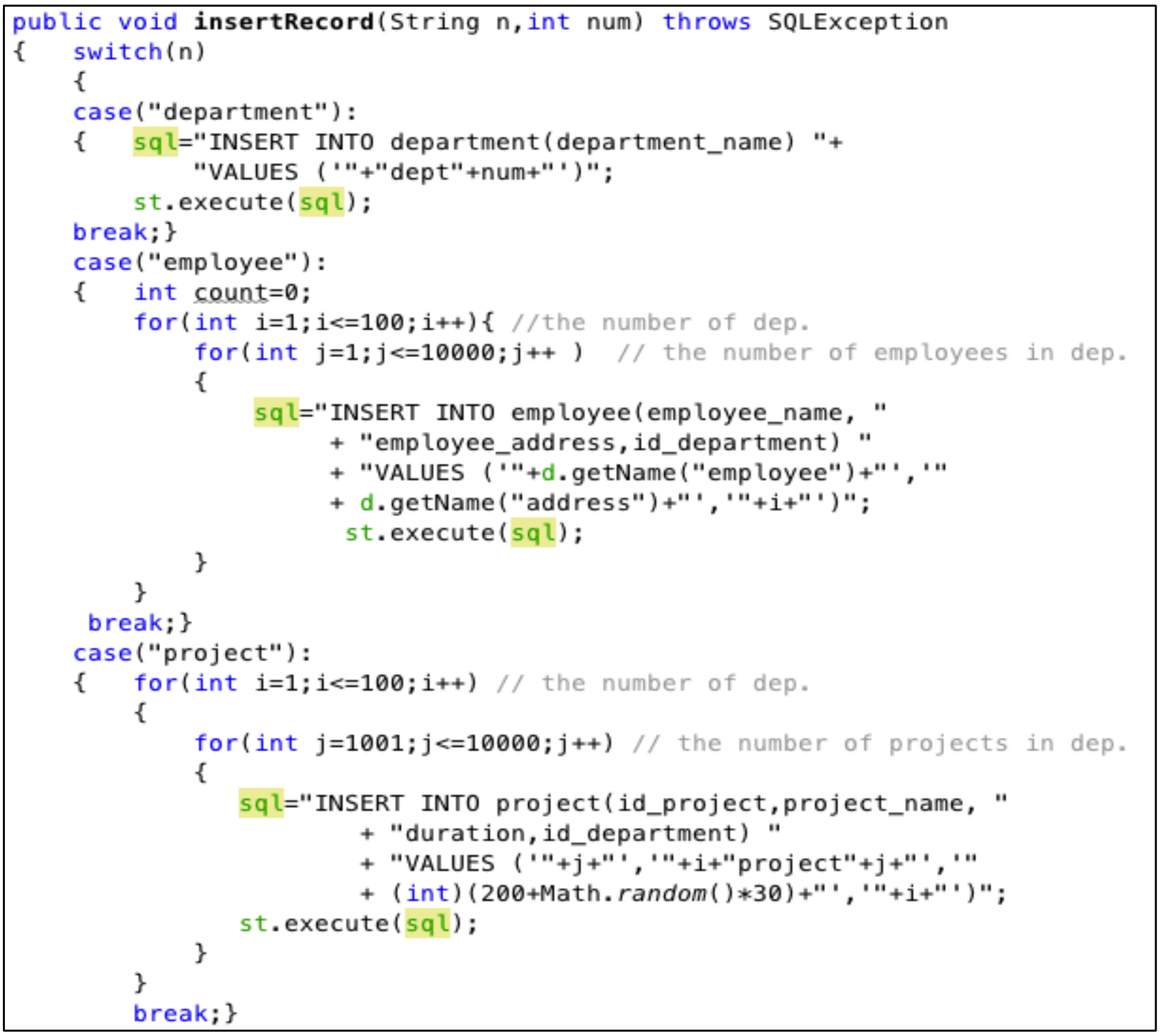
In this research, the focus will be on data retrieval only, i.e., data manipulation operations are not within the scope of this research.

The main aim of this research is to identify the best design structure of MongoDB that achieves the highest performance compared to Relational DB design.

1.2.2 Research methodology

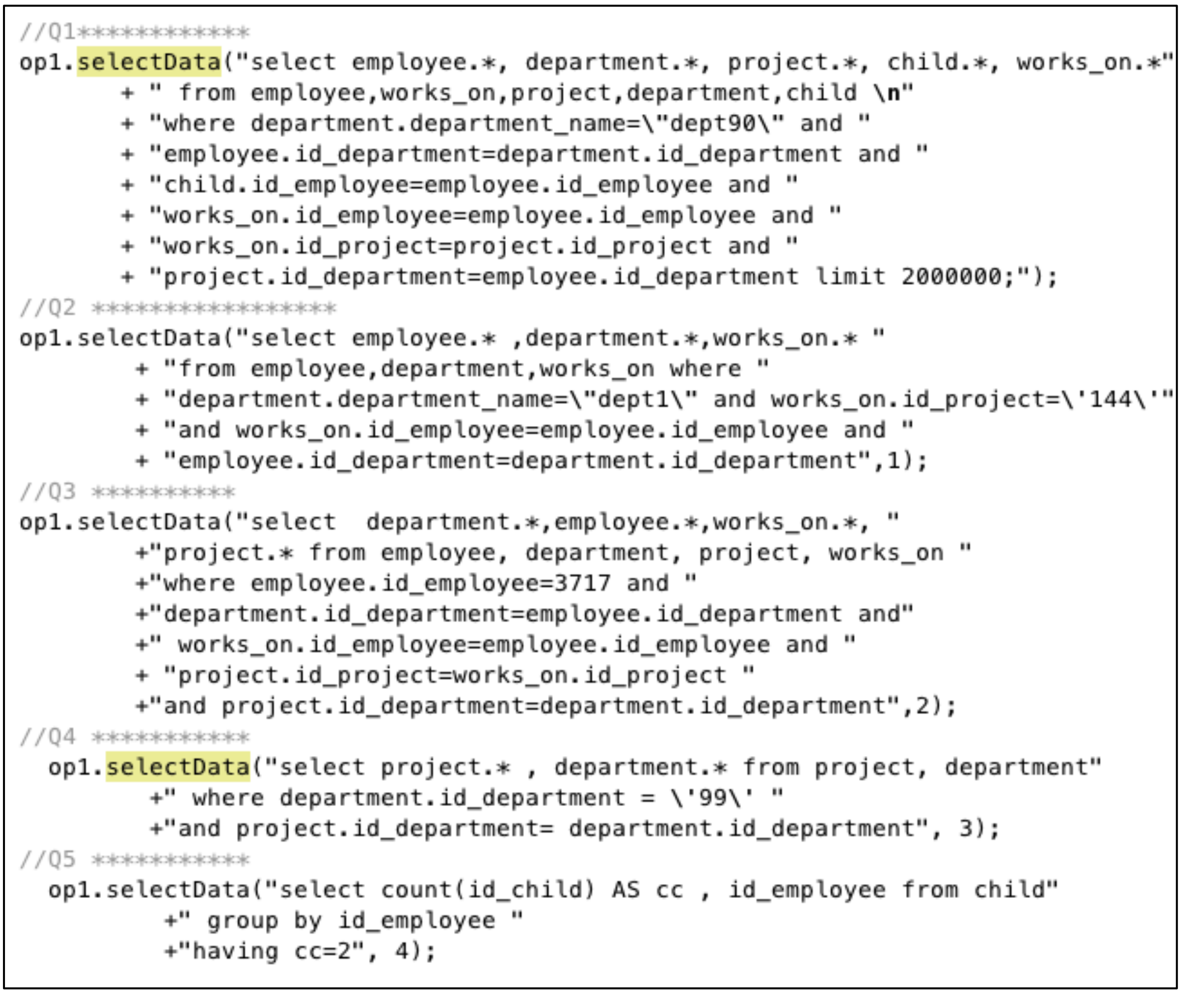
The basic methodology used in this research is to identify the problem and then manipulate solutions to this problem. The focus of this research went through the following parts:

i) Research input: MySQL server DB that contains 5 tables with one-to-many and many-to-many relationships between them. There is no one-to-one relationship in this experiment as implementing it in MongoDB will not add more embedding or reference relationships that affects the experiment result. Filling the tables with random data through java API see Figure 1. After that, applying five different types of queries to collect data from the tables based on the query conditions see Figure 2 and record the execution time for each query.



```
public void insertRecord(String n,int num) throws SQLException
{   switch(n)
    {
    case("department"):
    {   sql="INSERT INTO department(department_name) "+
            "VALUES ('"+"dept"+num+"')";
        st.execute(sql);
    break;}
    case("employee"):
    {   int count=0;
        for(int i=1;i<=100;i++){ //the number of dep.
            for(int j=1;j<=10000;j++ )  // the number of employees in dep.
            {
                sql="INSERT INTO employee(employee_name, "
                     + "employee_address,id_department) "
                     + "VALUES ('"+d.getName("employee")+"','"
                     + d.getName("address")+"','"+i+"')";
                      st.execute(sql);
            }
        }
     break;}
    case("project"):
    {   for(int i=1;i<=100;i++) // the number of dep.
        {
            for(int j=1001;j<=10000;j++) // the number of projects in dep.
            {
               sql="INSERT INTO project(id_project,project_name, "
                       + "duration,id_department) "
                       + "VALUES ('"+j+"','"+i+"project"+j+"','"
                       + (int)(200+Math.random()*30)+"','"+i+"')";
               st.execute(sql);
            }
        }
        break;}
```

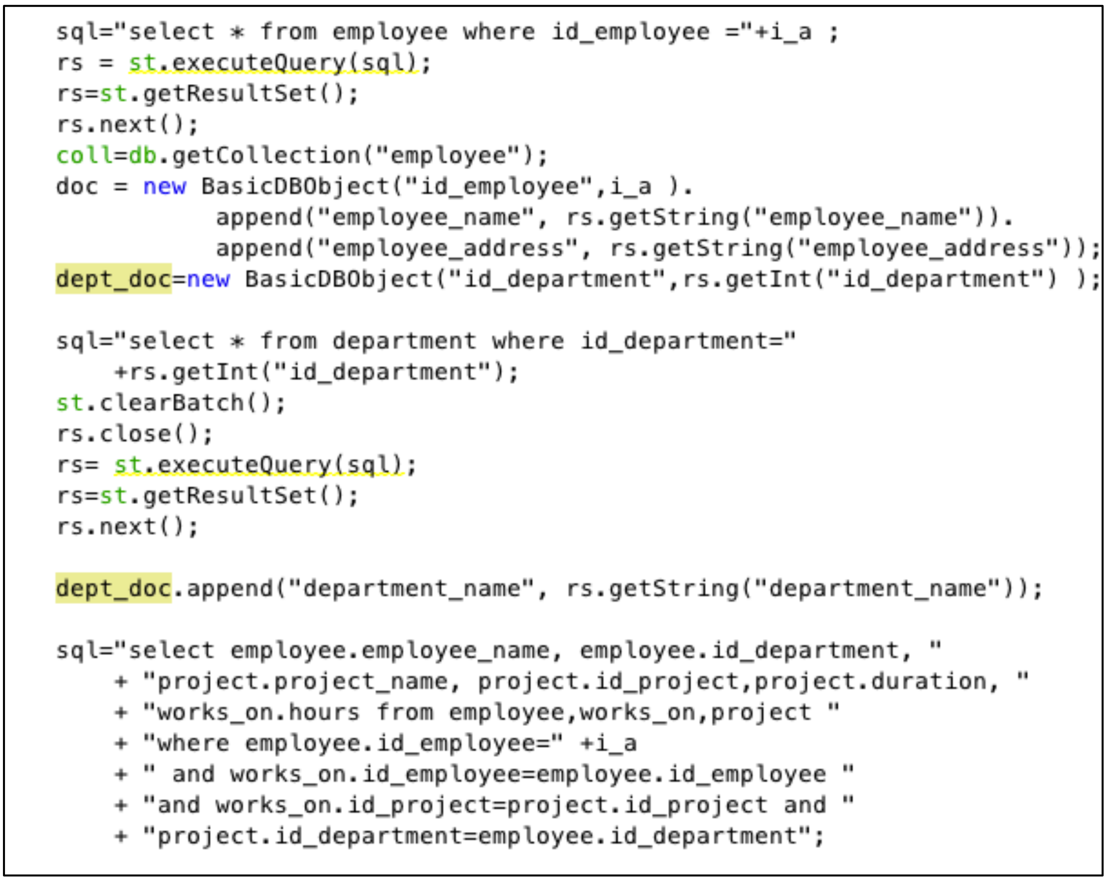
Figure 1: Insert data in MySQL database table - Java API



```
//Q1************
op1.selectData("select employee.*, department.*, project.*, child.*, works_on.*"
       + " from employee,works_on,project,department,child \n"
       + "where department.department_name=\"dept90\" and "
       + "employee.id_department=department.id_department and "
       + "child.id_employee=employee.id_employee and "
       + "works_on.id_employee=employee.id_employee and "
       + "works_on.id_project=project.id_project and "
       + "project.id_department=employee.id_department limit 2000000;");
//Q2 *****************
op1.selectData("select employee.* ,department.*,works_on.* "
        + "from employee,department,works_on where "
        + "department.department_name=\"dept1\" and works_on.id_project=\'144\'"
        + "and works_on.id_employee=employee.id_employee and "
        + "employee.id_department=department.id_department",1);
//Q3 **********
op1.selectData("select  department.*,employee.*,works_on.*, "
        +"project.* from employee, department, project, works_on "
        +"where employee.id_employee=3717 and "
        +"department.id_department=employee.id_department and"
        +" works_on.id_employee=employee.id_employee and "
        + "project.id_project=works_on.id_project "
        +"and project.id_department=department.id_department",2);
//Q4 ***********
  op1.selectData("select project.* , department.* from project, department"
         +" where department.id_department = \'99\' "
         +"and project.id_department= department.id_department", 3);
//Q5 ***********
  op1.selectData("select count(id_child) AS cc , id_employee from child"
          +" group by id_employee "
          +"having cc=2", 4);
```

Figure 2: Queries used in the implementation - Java API

ii) Research Process: NoSQL MongoDB is selected to create 3 different data structures that will be used to represent the same data stored in the MySQL server DB see Figure 3. The same queries used to collect data from MySQL server DB are used to retrieve data from each structure. The execution time for each query was recorded to preview the performance of each query.



```
sql="select * from employee where id_employee ="+i_a ;
rs = st.executeQuery(sql);
rs=st.getResultSet();
rs.next();
coll=db.getCollection("employee");
doc = new BasicDBObject("id_employee",i_a ).
          append("employee_name", rs.getString("employee_name")).
          append("employee_address", rs.getString("employee_address"));
dept_doc=new BasicDBObject("id_department",rs.getInt("id_department") );

sql="select * from department where id_department="
    +rs.getInt("id_department");
st.clearBatch();
rs.close();
rs= st.executeQuery(sql);
rs=st.getResultSet();
rs.next();

dept_doc.append("department_name", rs.getString("department_name"));

sql="select employee.employee_name, employee.id_department, "
    + "project.project_name, project.id_project,project.duration, "
    + "works_on.hours from employee,works_on,project "
    + "where employee.id_employee=" +i_a
    + " and works_on.id_employee=employee.id_employee "
    + "and works_on.id_project=project.id_project and "
    + "project.id_department=employee.id_department";
```

Figure 3: Filling MongoDB data - Java API

iii) Research output: Selecting the best Mongo representation that has the highest performance in retrieving data compared to other Mongo structures and the original MySQL DB.

1.3 Literature Review

In this section, we summarize and present the current state of the art regarding converting SQL database to NoSQL database.

Jia et al. developed a tool to transfer relational data model to NoSQL model specifically MongoDB and migrate data to a new structure [1]. They used the database log to assign description tags to entities and action tags to describe the relationships between entities. Based on the assigned tags, the tables and relationships will be either embedded or referenced in the MongoDB collection. They have tested their tool by choosing 3 tables and each table has only one-to-many relationship with other tables. The DB is small in terms of the number tables (only 3 tables), the kind of relationships between the tables (only one-to-many), and the complexity of queries (only 3 simple quires used that did not include aggregation functions). G. Zhao et al. [2] presented a tool for transforming SQL DB schema to NoSQL DB. This tool attained high performance for join queries, and contained a graph-transforming algorithm that offers a correct nesting sequence to generate nesting sequences among relational tables. In their proposal they mapped all the tables in the relational DB into MongoDB collections, and they offered a graph-transforming algorithm to generate nesting sequences among relational tables. After testing the proposal, the results showed high performance of the new data structure and high redundancy as well. Four tables with three foreign keys were used in the case tested and four different level join queries were used as well. They have not included any aggregation function in the query and the size of the database is not mentioned as well.

A new design of a database systems migration tool shared by G. Zhao et al.

[3], the design helped in converting relational database into HBase database. They mentioned that using aggregation in the new design will lead to duplication of information but this is not a concern as the storage is already available.

Converting a traditional information management system (MySQL) that is related to a school into a system that fully stores its data in NoSQL database systems is shown in the paper published by Z. Wei-ping et al. [4]. They concluded that MongoDB is faster than MySQL when more data is inserted in the database as well as the development process in MongoDB is faster than MySQL. They have used a small size database and conducted the performance test using two queries only.

More comparisons between the performance of the traditional database systems and NoSQL systems were made by A. Boicea, F. Radulescu and L. I. Agapin [5]. They found that oracle database is a good choice for small size data only while MongoDB is faster in inserting and deleting big size data.

J. S. van der Veen et al. [6] were trying to find out which is more suitable for storing sensors data in both physical servers and virtual servers SQL DB (PostgreSQL) or NoSQL DB (Cassandra and Mongo). They have concluded that the best database structure to be selected depends on the system requirements and the use of the sensors. The results discussed in the research shows that mongoDB readings was better in almost all the situations tested in the research. They did not perform the test on distributed systems and suggested adding more types of database to the comparision.

Based on the results of G. Zhao [7], W. Huang et al. investigated about the feasibility of the migration and potential performance of the system after the migration by modeling MongoDB with relational algebra, they set a certain assumption to convert the MongoDB to SQL DB. Then they applied the same relational algebra model used to define the relational database on the converted database and found that MongoDB supports relational calculus just like relational database. Therefore, the migration can be done safely and easily between the two data structures [7].

Researchers discussed adding a new layer between users and data that will enable the user to deal with different structures easily [9, 20]. R. Lawrence [9] suggested adding a unity layer between the data and the user where users can use SQL quiers to retrieve data from SQL or NoSQL strucutre using a single SQL query. Liao, Y. T. et al. [20] presented two types of data adaptors in addition to a database convertor tool. This system will provide non-stopping services while the data transformation is performed. It also avoids stopping the application and changeing its design before using the new NoSQL DB model. They have introduced 3 modes in their system: Blocking transformation mode, blocking dump mode, and direct access mode.

M. G. Jung et al. [12] assessed the performance of the relational and NoSQL systems, and provided optimal designs for best performance when using NoSQL. They tested the performance of PostgreSQL DB, MongoDB structured model (structured model like the PostgreSQL with 3 collections), and the unstructured MongoDB (one collection only). They showed that MongoDB with unstructured model have much better performance than PostgreSQL and better than using the MongoDB structured model (more than one collection).

In a study of comparing the performance of different database operations in both relational and non-relational databases for big data application (airlines data consistes of 1 million records and stored in three tables), S. Chickerur et al. [14] found that MongoDB is faster than the MySQL in inserting, selecting, updating and deleting data. In the article, they converted one table to MongoDB only, no relationships were shown in the database, the results in all the operation were close to each other even though the MongoDB showed better performance.

S. H. Aboutorabi et al. [15] compared the performance of the relational DB (Microsoft SQL server) and the NoSQL (MongoDB) when implementing them both in an e-commerce application, by testing all the operations read, insert, select, delete, aggregated and non-aggregated functions. Their results showed that Mongo DB achieved better performance in all of the tested operations except in the aggregated function test, and MongoDB needs more focus with non-indexed data. They displayed a big ERD for the database used but they have not mentioned if they included all the tables in the experiment.

A framework that enables representing the relational DB of running applications as NoSQL with minimum human effort and less time had been presented by L. Stanescu et al. [17]. They listed a set of rules to convert different types of relational databases into MongoDB based on the table constraints in the SQL information schema. They also mentioned the benefits of MongoDB that puts it furthur ahead of the relational database.

P. Gómez, R et al. [19] argued that structuring data has a great impact on data size, query performance, and code readability which indeed affects the program debugging and maintenance. They compared different models (structures of data), different embedding structures, different access patterns, and they used indexes as well. The experiment was done using MongoDB and an evaluation of the findings was also discussed. In their research, they concentrated on the implementation of a single 1:M relationship between two tables. Extending this expermint to test more complex relationships and tables involoved in more than one type of relationship with other tables will enrich the results of this research.

This reseach is different from the previous work in considering bigger scale of study in term of number of tables, more improtantly vairous relationship between tables that covers major relationships, namely, one-to-many, and many-to-many and the amount of data stored in the database.

In this research, a relational database was redesigned to have a new structure of NoSQL database specifically MongoDB that has high performance, low redundancy and reliable taking into consideration the special characteristics of MongoDB. We worked in one complete structure of relational database that includes 5 different tables with different types of relationships (one-to-many and many-to-many) between them. One of the tables has 3 relationships with three different tables. Another one has 2 relationships with two different tables in the same database. We have converted the relational database from MySQL into three different structures of the NoSQL MongoDB, the first structure has single document to represent the data, while in the second includes references and embedding documents and the last one considered a collection for each table and created reference relationships between them.

1.4 Relational Database Structure

Relational database has been used long time ago to represent data; it has been used for the past 30 years [9] and it is still one of the most widely used structures to represent data [8] because developers are familiar with it [6], its applicable to wide variety of data problems, and variety of vendors are available which gives the customer more flexibility regarding the cost, feature and performance [9]. Despite that there are several new structures that are competing in the market, relational model is theoretically grounded and efficient to implement [9].

Relational DB strength comes form its architecture that is based on physically representing data using fixed table structure [6, 7], interrelated tables [3], two-dimensional tables [4], and views as virtual representation of the relationships between the tables [7]. Primary keys, indexes are important parts of the relational DB structures [14] in addition to foreign keys that are used to link tables with each other [6, 14]. Quires written in SQL language are used to retrieve and manage data in relational DB [6, 7]; those queries can vary from being very simple accessing one table to complex where many tables are involved in what is known as join quires [3] or join operator [6]. It offers normalization in different forms [3, 14] and enforces data integrity as well [10, 14]. Furthermore, it is small in size, fast, cheap [4] in terms of performance with small amount of data [10]. The data represented in relational DB is strongly stable, consistence, and available [6].

As data recently is growing rapidly and very fast, relational DB is facing many challenges like less ability to scale, less efficiency, in addition to the restrictions of the ACID [3] when data consistency became less important in the new systems requirements [7]. Relational DB is not very effective to represent huge amount of data [3] because of the low querying efficiency [7, 21] of multi join that is used in big data [4]. Adding to the previously mentioned restrictions on using relational database with big data, relational DB lacks the properties of reliability [2], distribution [8] and scalability [2, 7]. Regarding scalability, it is difficult to distribute the SQL database horizontally but scaling the data vertically can be easily achieved by upgrading the database server [6]. Applications are portable and can be moved to other system, with some changes to be made for some procedures and system specific features [9].

Object-relation inconsistency is one of the weak points of relational DB, which means that the relational model is different than the data structure in the memory and this does not also make relational data base a good solution for representing big data [12] as well as high maintenance cost [7].

Relational DB has high latency time that prevents it from being used for real-time data storage [8]. The normalization and indexing require extra tables to be added to the relational DB [14, 15] and this will result in more joins, keys and indexes. As a result, several issues like requiring more space, and low performance of the database will appear. Upgrading the hardware is a good solution to those issues but this is not sufficient as it will be expensive for storing the data, support and maintenance [14].

1.5 Object Oriented Database Structure

OODB can be defined as representing data as objects. As the OO structure is used there is classes for the objects, inheritance, methods and subclasses. This kind of DB has a management system known as OODBMS that supports all the functions related to the data representation as OO [23].

OODB has been widely used in telecommunications, transportation, and building management for years. When choosing to use OOP to deal with data in the database, the OODB will be the best choice rather than the RDB. OOBD is fully integrated with OOP as both of them are using the same objects concept, and this integration makes the connection and communicating with data easier and faster [24].

1.6 NoSQL Database Structure

NoSQL is the new paradigm of representing data that emerged to fulfill the need of high performance query, high concurrency, low latency among huge data [3, 4] and high speed [18]. MongoDB is an open source [9, 21] database management system that makes processing of massive and /or unstructured data easier [12]. It has a database server and a simple query API for querying the data in the database instance; its non-fixed schema is referred to as a dynamic schema [21].

It was developed to support applications that are not well served by relational DB [9] especially with many web applications available [3, 4] that relational DB failed to achieve [6] like the internet, social media, multimedia [10, 21], streams, and for big data processing [9] as it operates well with clusters [12].

NoSQL has lots of features that make it totally different from relational DB. NoSQL is a non-relational [4, 12], schema less [3], handle unstructured and different types of data [12] and efficiently process it [21]. Unstructured data is defined as information that is organized in a predefined manner without a predefined data model like body of the email, blobs, audio and video [21]. Also it has unstructured ways to store and retrieve data [2, 4, 17]. It does not support the join query [2, 4] and has less powerful query language to retrieve data than the relational DB [6]. In other words, it is better to say that NoSQL has no common query language available so it needs custom API to interact with the system using NoSQL DB to be able to communicate with the data [9]. NoSQL solved the mismatch between the relational DB and the OOP [17].

NoSQL uses key value format to store data [6, 10]. It also performs fast read and write because of the map functions used in processing big data [12] as well as high flexibility in adding or deleting attributes [10].

NoSQL is very flexible, reliable [5, 18], its structure is more based on what a you are doing of data, and does not need fixed tables to store data [5]. Basically NoSQL relaxes either consistency or availability of data that is very helpful to distribute data across networks [6]. NoSQL is not restricted by ACID and this is one of the main reasons of its high performance, high scalability [7, 9, 18] and high availability [7, 18]. On the other hand, it offers BASE properties. BASE is an acronym coined by Eric Brewer who developed the CAP theorem about consistency, availability and partition-tolerance [21]. It is suitable to be used in systems that deal with short data inconsistency and location independence [18]. It supports distributed data mining [2] and horizontal scalability [21].

Definitely, easy design and implementation, high performance and horizontal scalability are of the strength points of the NoSQL DB [6]. Although NoSQL needs huge storage [6] but it is not meant to be a concern as storage is available and cheap over the cloud [3, 18, 21].

NoSQL has four different ways to represent data. Designers select one of them based on the enterprise requirements. Those representations are [9, 10]:

- Key-value stores like HBase: uses hash interface to store and retrieve data through a simple interface [9]. The key can be self generated and the value can be anything [2, 7, 10, 15, 18, 19].
- Document stores like MongoDB, CouchDB: it attaches structured documents with a key, it has different representation format like MongoDB is using BSON [9]. The document types of NoSQL DB are key-value database with the ability to find documents based on their contents [2, 7, 10, 15, 18, 19].
- Column stores comes from the BigTable category. HyperTable is one example of NoSQL databases using the columns stores. This type of data representations requires a predefined schema. Data is saved in cells, cells grouped in columns, and columns grouped in families. The columns can be created at run time or using predefined schema [2, 7, 10, 18, 19].
- Graph database like Neo4J [2, 7, 9, 10, 18, 19].

Despite all the solutions provided by NoSQL, it has few drawbacks; for example, using aggregation may lead to duplication of information [3]. As it has no standard way to access data, it requires system-specific code to do that, which reduces the adoption of the new system [9]. Inability to structure unstructured data is one of the main issues of NoSQL, as well as the high performance cost of processing big data [12].

Lots of examples of NoSQL DBs available like BigTable used by google and it was the first NoSQL DB [15, 21], MongoDB, HBase, Cassandra [20], Facebook

Cassandra, Amazons' SimpleDB, Microsoft Azure, and Oracle Corporations' Oracle NoSQL [21].

1.6.1 Converting relational DB to NoSQL DB.

Many applications started to adopt MongoDB instead of their old relational DB [5] like Telefonica [1]. This shift towards NoSQL DB is facing many challenges such as schema conversion and maintaining the reading efficiency after the conversion [2]. Some organizations prefer to keep the old relational DB and use new NoSQL DB, hence; they will end up with running two DB implementations at the same time [20].

One of the data model transformation challenges is that, it is done most of the time manually by experts. The expert should consider that MongoDB does not support join and when to embed or reference tables. Such critical decision might affect the performance and data redundancy of the DB. Regarding data migration, the expert should move all the data into the new model correctly [1].

Data migration from relational DB to NoSQL is not an easy task to accomplish because of the absence of the methods that guide the migration. After the migration, there is no evaluation on the performance and the capabilities of the data in the new data model compared to the data in the old model [7].

1.6.2 Why to choose MongoDB

MongoDB is one type of the NoSQL databases that stores data in a structured way as JSON like document called BSON [10, 19]. It is developed in C++ [7, 15, 21]. It is best described as dynamic representation of data that makes implementing it easier and simpler on most types of applications [5, 14]. MongoDB was launched in

2009 [1, 5, 10, 15], and it is still expanding and developing [10]. One of the main reasons of MongoDB popularity is the focus on the flexibility, speed, power and ease of use [5].

The need of MongoDB emerged as a result of the failure of relational DB to handle applications with very large datasets and very flexible data structure [1, 19]. Many applications started to adopt MongoDB instead of their old relational DB [5] like Telefonica [1].

MongoDB has become a good solution for the new applications as it does not require predetermined data schema. It is an open source and a document-oriented DB [1, 4, 5, 6, 7, 15, 17, 19, 21] that can store different types of objects like XML, JSON, BSON and other types [1, 5, 15, 19, 21]. The ACID transaction properties are not considered and they are replaced with the BASE architecture [1, 5, 15, 19]. Join and transaction concepts are not introduced in MongoDB [13] that help in improving its performance [1, 4]. It is a cross platform DBMS and supports multiusers [5]. Despite that it is new in the market, it proved high functionality. Lots of big companies have their own justification to choose using MongoDB in their projects [5, 15]. Some of the major attractive features of MongoDB include scalability [13, 15, 17]; to meet the web2.0 applications [7]; usability in distributed environment [6, 15]; that is suitable for real time query data and massive log analysis [8]; high performance [7, 13, 15], load balancing [7] as it automatically sets data to portioning mode and this helps in dividing the load evenly and improves the performance [15, 17], easy way to store data [15], consistency, durability, conditional atomicity [5], and availability [13, 17].

MongoDB has lots of features that make it a preferred solution for lots of companies. It supports dynamic and non-predefined schemas [1, 5, 4, 7, 15, 19], ability to be used with small size project and big data projects [6, 10]. It supports serialization, indexes [13], map/reduce operations, master-slaves replication [7], and data sharding that are important for achieving horizontal scalability and high availability of data [1, 5, 19]. Including indexes in MongoDB is an option available for the database users [6, 7]. Using indexes decrease the data read time in both virtual and physical server and help in locating data easily [6], because it stores index into memory and leaves data on disks [8]. It also uses internal memory for storing the working set to enable faster access to data [17].

MongoDB is made of collections that include documents. Documents contain simple and complex structures like lists, arrays, documents, etc. [4, 10, 19, 17, 21] with different data types content. Documents are structured as “field: value” [19] or “key: value” [6]. Each document has an ID field [10] that is given automatically or assigned by the user [1, 5]. It does not support the join but it has the reference and embedding features [3, 7, 10, 17, 19]. Embedding means adding a document inside another document. Reference means adding one or more fields of a document in another document [19]. MongoDB includes rich data processing functions [13] like creating and dropping a collection. When inserting the first record into your database, the collection will be created automatically. The absence of dependences between collections allowed safely deleting collections [5]. Inserting new data has no constraints and is achieved by using one of the functions save or insert. More functions are available to find data use the function find(), sort data use the function sort(), remove data use the function remove(), and update data use the function update() [5].

MonogDB provides rich document-based query language [13] that is applied to a concrete document collection. The complexity of the query is related to the number of collections involved, and the embedding level in the document. Filters, projections, selections [19], aggregation [7] and many other operators to compare and find data in a document exist in the language supported by the database [19]. The aggregation operations for example can be divided into several phases including $project, $wind, $unwind, $group, $match [8]. Developer's skills are very important to improve the performance and readability of the query program [19]. Also developers have flexibility to choose any programming language to use with MongoDB [5, 9, 15] because it consists of API calls, java scripts [6]; as the JSON objects stored in MongoDB can be easily converted to javascripts objects in code [9]; and REST to query data [5]. MongoDB provides well query performance and aggregation analysis [8].

There is no limit to embedding too many documents in one document, but this process will increase the document size and will slow down the query [4].

Chapter 2: Experiment Implementation

2.1 Data Structuring

This section explores the different database structures used in the research. First, we will explain in details the relational DB representation and then we will go through the three different designs of the same data in Mongo database.

2.1.1 MySQL server DB structure

The sample DB used in the research is named Employee, which ERD is shown in Figure 1. The ERD shows that the DB contains 5 tables with multiple relationships and different types of relationships. Each department has many employees and runs many projects. Every employee in a department might be involved in more than one project in her/his department only. The many_to_many relationship between project table and employee table is represented in the table works_on. Employee table has a one_to_many relationship with child table. Primary and foreign keys are indexes in all the tables. The relationships and number of records inserted in each table are summarized in Table 1.

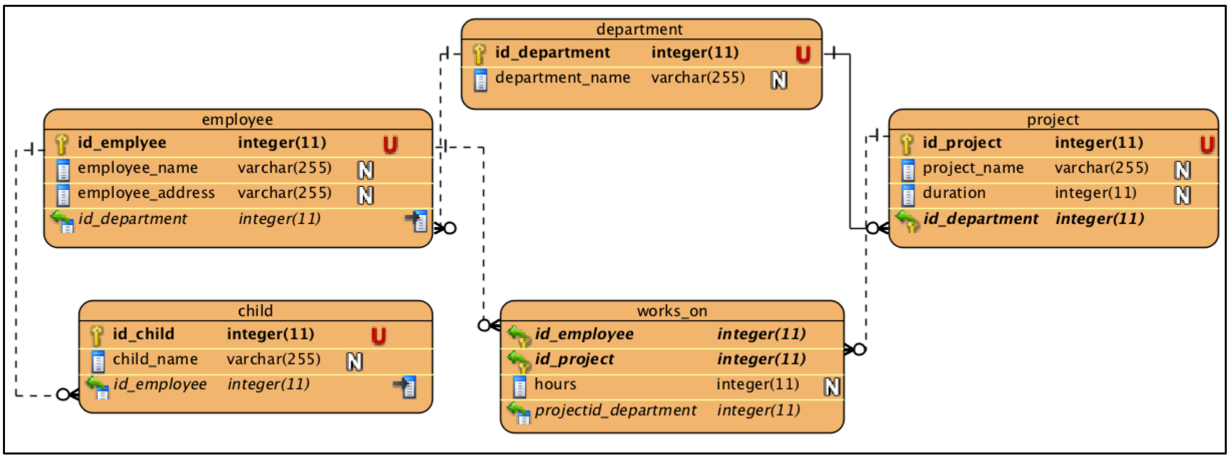

department
id_department integer(11) U
department_name varchar(255) N
employee
id_emplyee integer(11) U
employee_name varchar(255) N
employee_address varchar(255) N
id_department integer(11)
project
id_project integer(11) U
project_name varchar(255) N
duration integer(11) N
id_department integer(11)
child
id_child integer(11) U
child_name varchar(255) N
id_employee integer(11)
works_on
id_employee integer(11)
id_project integer(11)
hours integer(11) N
projectid_department integer(11)

Figure 4: ERD for employee DB

Table name	Type of relationship	Relationship with table	Number of records
Employee	Many to one	Department	2 million
	Many to many	Works_on	
	One to many	Child	
Department	One to many	Project	100 records
	One to many	Employee	
Project	Many to one	Department	2 million
	Many to many	Works_on	
Child	Many to one	Employee	40 millions record
Works_on	Many to many	Employee	11 millions record
	Many to many	Project	

Table 1: MySQL tables details

The database was created through a java API and filled with random data using the same API.

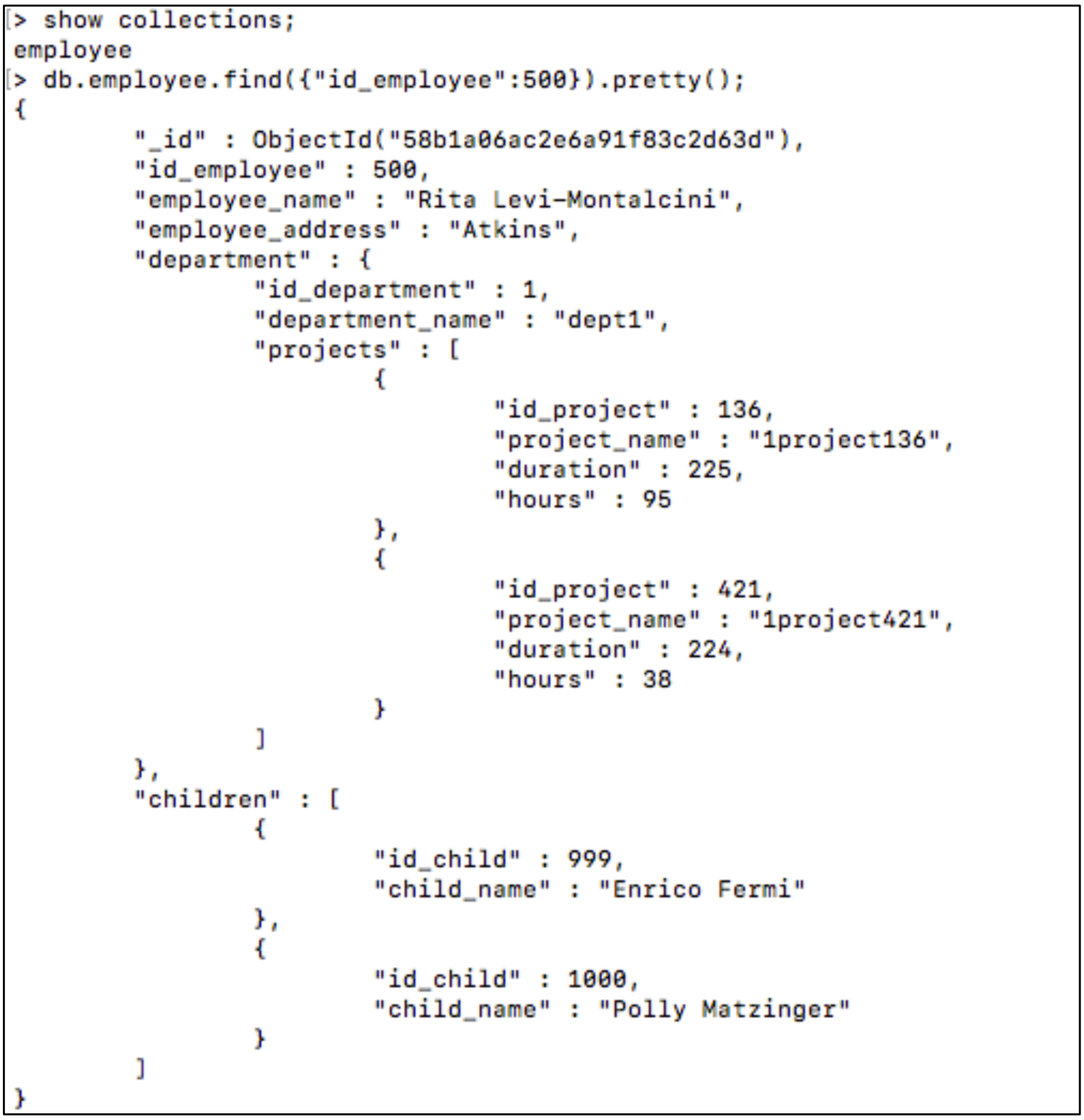
2.1.2 Mongo DB structures

As MongoDB contains two types of relationships between their collections: embedded and reference relationships, both of them are used in this research. Three database structures were designed and will be explained in details in the coming section. Different combinations of the those relationships are tested to find out which is the best way to achieve best results of querying data in the term of time efficiency. The first structure uses one collection. Data is represented using embedded relationship. The second one has two collections with both embedded and reference relationships. And the last one contains reference relationships between all the collections.

The database was created using the terminal window but creating all the collections and filling them with data was through java API. It is important to mention that the data used in all the collections is identical to the data stored in MySQL tables.

2.1.2.1 Mongo Structure 1 (Fully embedded document)

After studying the MySQL database design, we found that the best way to represent all the data in one document is to embed all the data related to one employee in a document called Employee. Employee document contains all the information related to the employee. The department information in which she/he is working is represented as an embedded document. The project details that the employee is involved in are the second embedded document and her/his children information as the third embedded document in the collection. Figure 5 shows sample of the document employee, it shows a clear picture about the embedded documents. The collection contains 2 million employee documents.

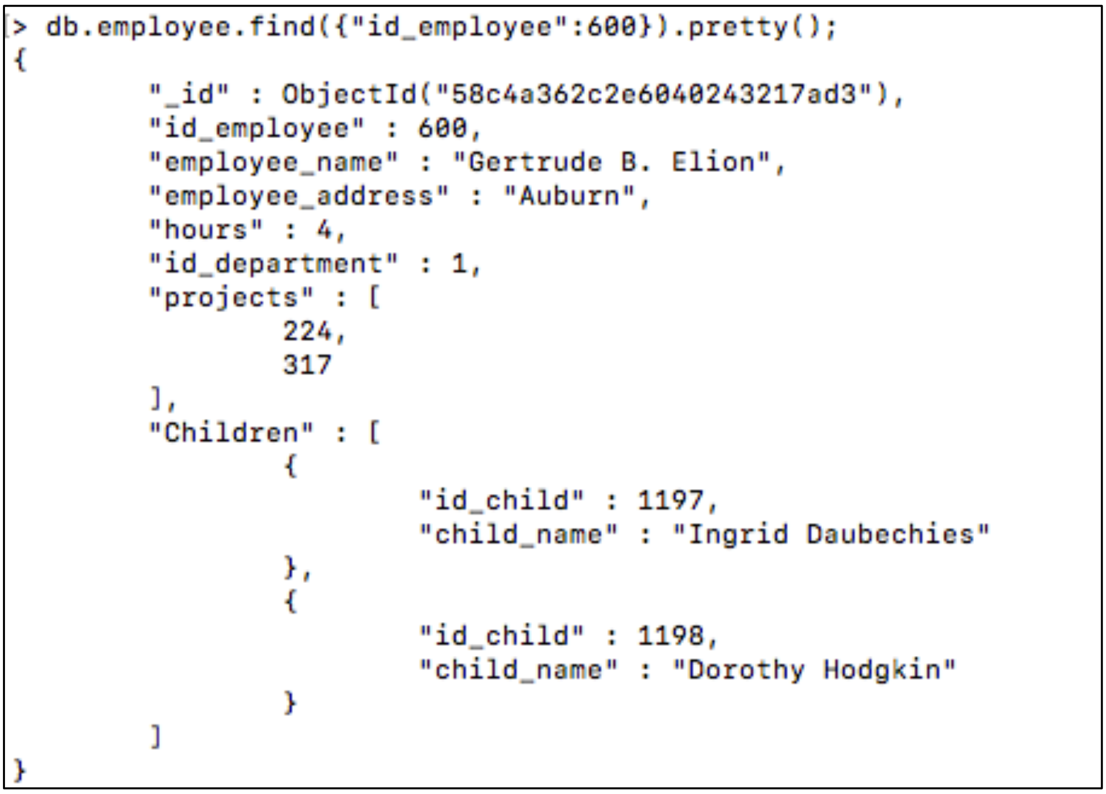


```
[> show collections;
employee
[> db.employee.find({"id_employee":500}).pretty();
{
        "_id" : ObjectId("58b1a06ac2e6a91f83c2d63d"),
        "id_employee" : 500,
        "employee_name" : "Rita Levi-Montalcini",
        "employee_address" : "Atkins",
        "department" : {
                "id_department" : 1,
                "department_name" : "dept1",
                "projects" : [
                        {
                                "id_project" : 136,
                                "project_name" : "1project136",
                                "duration" : 225,
                                "hours" : 95
                        },
                        {
                                "id_project" : 421,
                                "project_name" : "1project421",
                                "duration" : 224,
                                "hours" : 38
                        }
                ]
        },
        "children" : [
                {
                        "id_child" : 999,
                        "child_name" : "Enrico Fermi"
                },
                {
                        "id_child" : 1000,
                        "child_name" : "Polly Matzinger"
                }
        ]
}
```

Figure 5: Employee document sample data

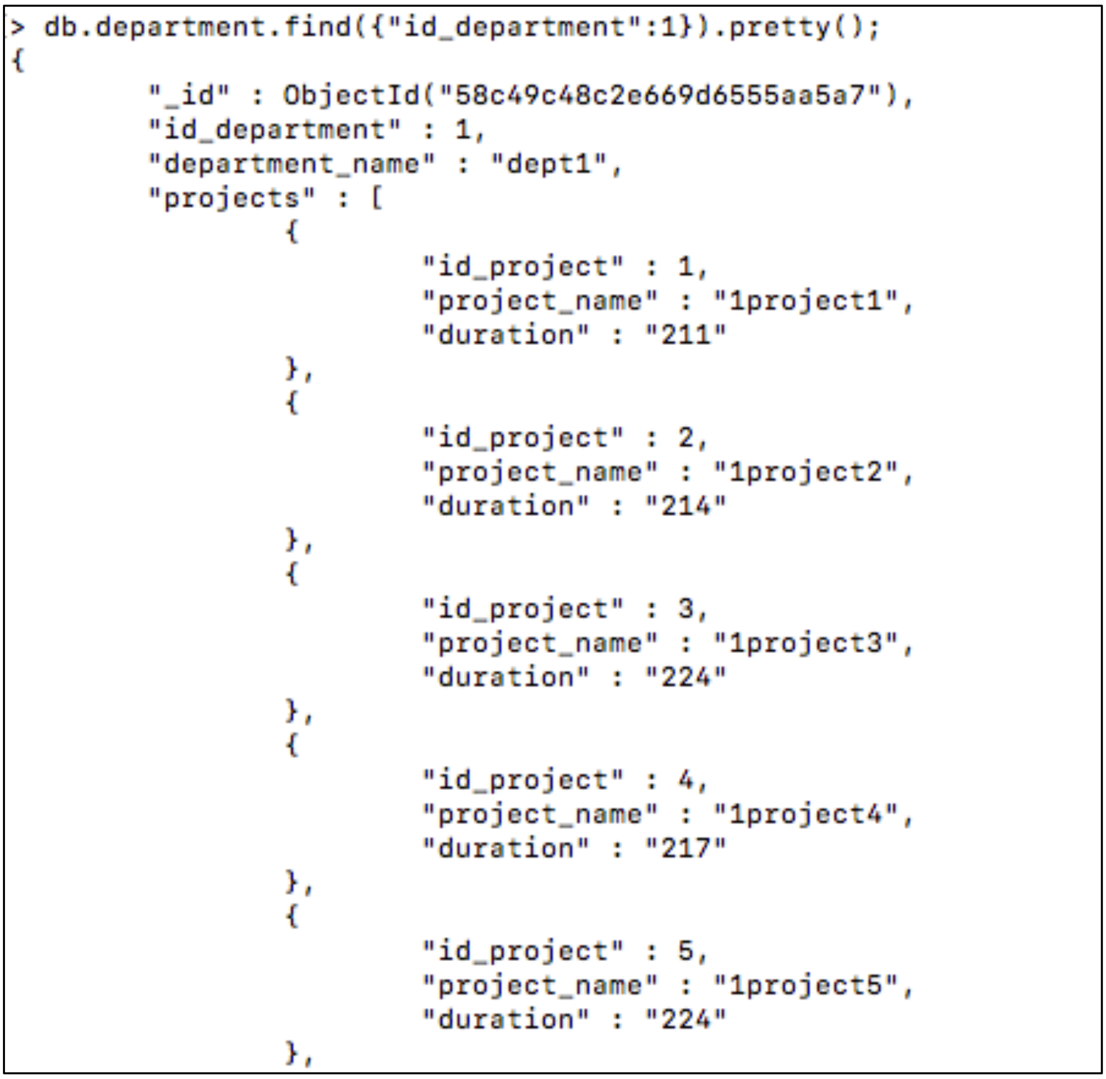
2.1.2.2 Mongo Structure 2 (Embedded and reference documents)

This representation divides the data into two documents. The first document is the Department document, and it contains information about departments and their projects. Project information is represented as an embedded document in the department collection. The second document is the employee document that contains information about the employee, its department using the reference relationships, projects she/he is involved in as a reference relationship as well, and the children information as an embedded document. Figure 6 shows a sample of the documents department and employee; it shows a clear picture about the embedded documents and reference relationships. The department collection contains 100 documents and the employee document has 2 million documents.



```
> db.employee.find({"id_employee":600}).pretty();
{
        "_id" : ObjectId("58c4a362c2e6040243217ad3"),
        "id_employee" : 600,
        "employee_name" : "Gertrude B. Elion",
        "employee_address" : "Auburn",
        "hours" : 4,
        "id_department" : 1,
        "projects" : [
                224,
                317
        ],
        "Children" : [
                {
                        "id_child" : 1197,
                        "child_name" : "Ingrid Daubechies"
                },
                {
                        "id_child" : 1198,
                        "child_name" : "Dorothy Hodgkin"
                }
        ]
}
```

Figure 6: Department and Employee documents sample data

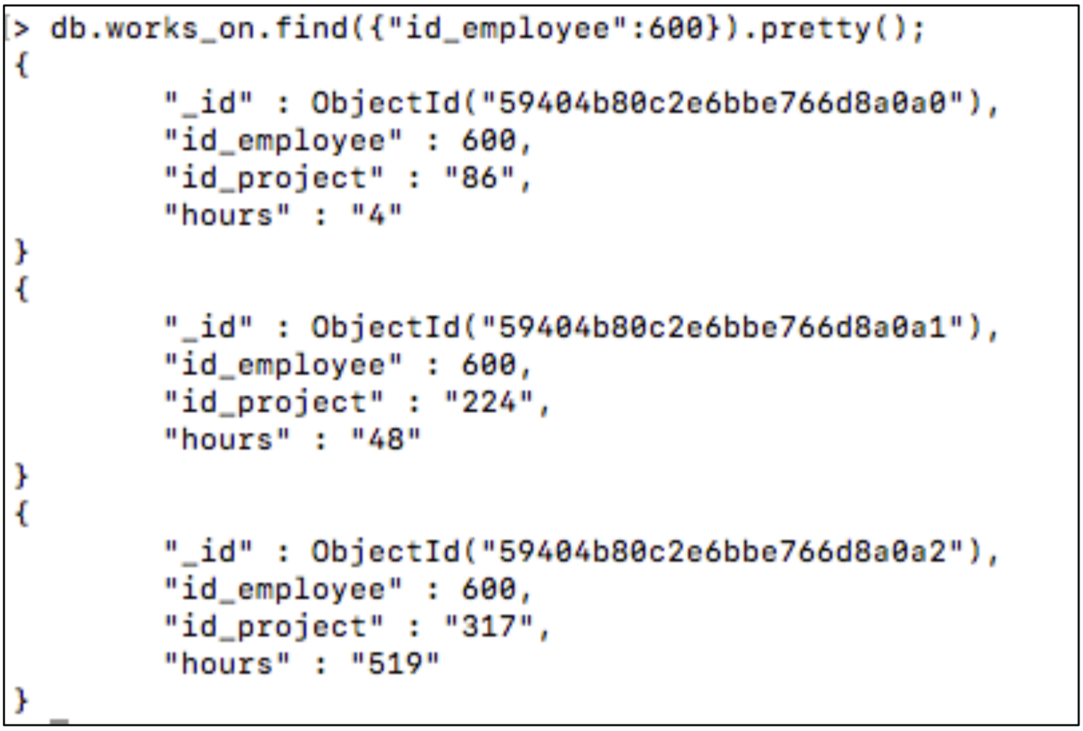


```
> db.department.find({"id_department":1}).pretty();
{
        "_id" : ObjectId("58c49c48c2e669d6555aa5a7"),
        "id_department" : 1,
        "department_name" : "dept1",
        "projects" : [
                {
                        "id_project" : 1,
                        "project_name" : "1project1",
                        "duration" : "211"
                },
                {
                        "id_project" : 2,
                        "project_name" : "1project2",
                        "duration" : "214"
                },
                {
                        "id_project" : 3,
                        "project_name" : "1project3",
                        "duration" : "224"
                },
                {
                        "id_project" : 4,
                        "project_name" : "1project4",
                        "duration" : "217"
                },
                {
                        "id_project" : 5,
                        "project_name" : "1project5",
                        "duration" : "224"
                },
```

Figure 6: Department and Employee documents sample data (cont.)

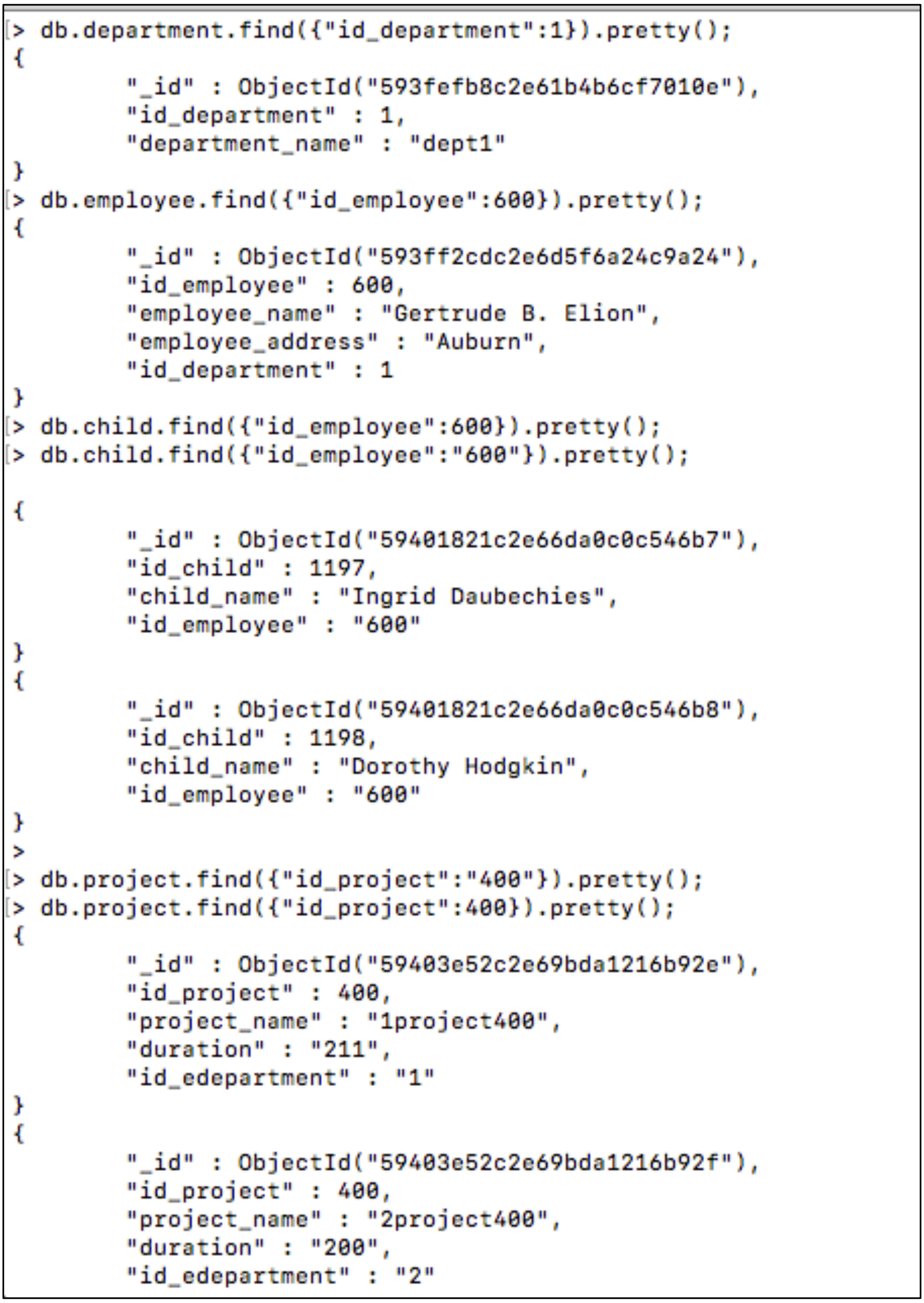
2.1.2.3 Mongo Structure 3 (All Reference documents)

This representation implements each table in MySQL database in a separate collection. Each collection has a reference relationship with the other collections. The first document, Department document doesn’t have reference relationship with any other collection. The employee collection has a reference relationship with department collection. The project collection has a reference relationship with department collection. Child document has a reference relationship with employee collection. The last collection works_on has a reference relationship with both employee and project collections. Figure 7 shows a sample of the how data is referenced in all of the five documents.



```
> db.works_on.find({"id_employee":600}).pretty();
{
        "_id" : ObjectId("59404b80c2e6bbe766d8a0a0"),
        "id_employee" : 600,
        "id_project" : "86",
        "hours" : "4"
}
{
        "_id" : ObjectId("59404b80c2e6bbe766d8a0a1"),
        "id_employee" : 600,
        "id_project" : "224",
        "hours" : "48"
}
{
        "_id" : ObjectId("59404b80c2e6bbe766d8a0a2"),
        "id_employee" : 600,
        "id_project" : "317",
        "hours" : "519"
}
```

Figure 7: Reference relationship between 5 documents sample data



```
[> db.department.find({"id_department":1}).pretty();
{
        "_id" : ObjectId("593fefb8c2e61b4b6cf7010e"),
        "id_department" : 1,
        "department_name" : "dept1"
}
[> db.employee.find({"id_employee":600}).pretty();
{
        "_id" : ObjectId("593ff2cdc2e6d5f6a24c9a24"),
        "id_employee" : 600,
        "employee_name" : "Gertrude B. Elion",
        "employee_address" : "Auburn",
        "id_department" : 1
}
[> db.child.find({"id_employee":600}).pretty();
[> db.child.find({"id_employee":"600"}).pretty();

{
        "_id" : ObjectId("59401821c2e66da0c0c546b7"),
        "id_child" : 1197,
        "child_name" : "Ingrid Daubechies",
        "id_employee" : "600"
}
{
        "_id" : ObjectId("59401821c2e66da0c0c546b8"),
        "id_child" : 1198,
        "child_name" : "Dorothy Hodgkin",
        "id_employee" : "600"
}
>
[> db.project.find({"id_project":"400"}).pretty();
[> db.project.find({"id_project":400}).pretty();
{
        "_id" : ObjectId("59403e52c2e69bda1216b92e"),
        "id_project" : 400,
        "project_name" : "1project400",
        "duration" : "211",
        "id_edepartment" : "1"
}
{
        "_id" : ObjectId("59403e52c2e69bda1216b92f"),
        "id_project" : 400,
        "project_name" : "2project400",
        "duration" : "200",
        "id_edepartment" : "2"
```

Figure 7: Reference relationship between 5 documents sample data (cont.)

Table 2 summarizes the collections used and number of documents in each one of the mongo structures explained earlier.

Collection name	Mongo Structure 1	Mongo Structure 2	Mongo Structure 3
Employee	2 millions	2 millions	2 millions
Department	-	100 documents	100 documents
Project	-	-	1 million
Child	-	-	40 millions document
Works_on	-	-	5 millions document

Table 2: Mongo database structures details

2.2 Query Description

After creating the databases in both MySQL and Mongo and filling it with data, five different types of queries were used to retrieve data. Different types of queries were included with different levels of difficulty, the number of tables and collections involved in retrieving the data and aggregation functions are used as well. The 5 queries are explained in Table 3.

Query Number	Description
1	• Select all information about the employees that work in a certain department. • Each department in the experiment has 20000 employees.
2	• Select all information about the employees working in a certain project in a certain department. • The employee can work on different projects in his department only.
3	• Find all the information about the projects where an employee is working.
4	• Select all information about the projects in a certain department. • Each department in the experiment has 20000 projects.
5	• Select the employees who have a certain number of children. • Each employee has a maximum of 3 children.

Table 3: Queries description

To get more accurate results, the average execution time of running the same query 20 times on the same dataset was calculated, and to make be fair in calculating the execution time in all the structures, no additional indexing for any structure were added. MySQL database has primary key and foreign key as indexing fields and mongo DB has the collection id as an index only.

Chapter 3: Results and Discussion

3.1 Data Query Results based on Database Structure

In this section, we will elaborate on the results that we obtained and justify/explain the results.

3.1.1 Results of queries on MySQL database

The chart in Figure 8 below shows the time spent in executing the 5 queries explained earlier. The chart shows that the time needed to retrieve all the data related to one employee is very short because of using primary or foreign keys in each table; it is about 1.10 parts of the second. While the last query that requires aggregation took very long time 299.37 parts of the second, this result is accepted as the count aggregation function for each employee is processed then a check to find if it meets the condition given or not is executed. The three queries in the middle have more conditions to be satisfied before retrieving data; the time needed to execute those queries was very close to each other in average of about 67 parts of the second.

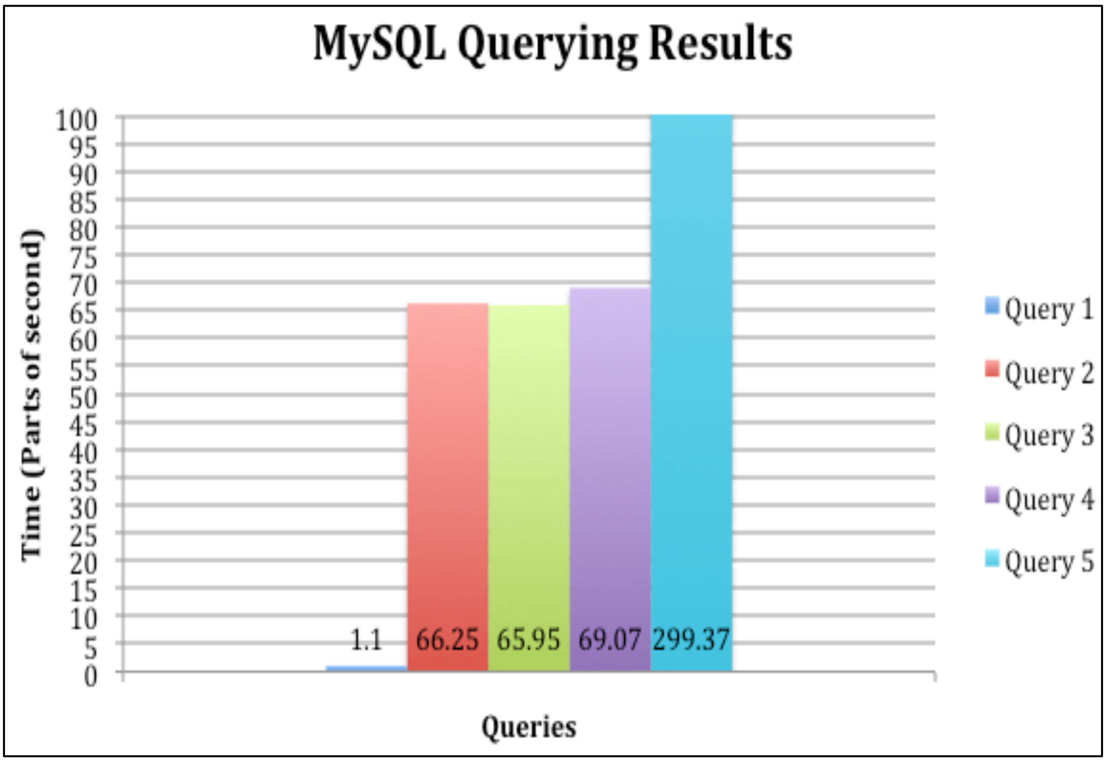


Figure 8: MySQL query results

3.1.2 Results of queries on Mongo database (Structure 1)

The chart in Figure 9 below depicts the time consumed in executing each one of the 5 queries used in the research when data is presented in one Mongo collection with embedded documents. The 3 embedded documents in the collection contain the data related to the department, projects and children. Department and children are in the same embedding level -first embedding level- and project is embedded in the department document-second embedding level-.

About 2 parts of the second needed to execute the first 3 queries and the last one as well. But for the fourth query where the data requested will be collected from the embedded documents only, as the query will pass all the second embedding level documents (project) to find all the projects related to each department (refer to table 3), it took longer time compared to the other queries.

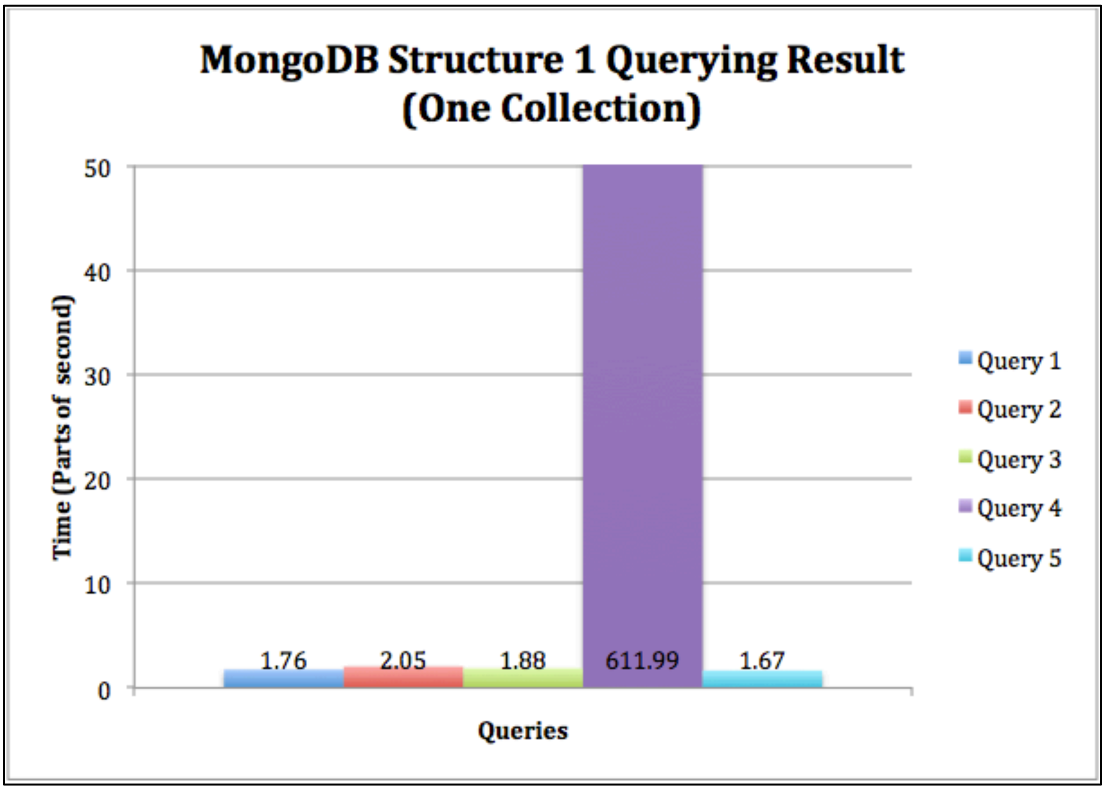


Figure 9: MongoDB Structure 1 - query results

3.1.3 Results of queries on Mongo database (Structure 2)

The chart in Figure 10 below illustrates the time consumed in executing each one of the 5 queries used in the research when data is presented in two Mongo collection with embedded documents and reference documents. There is one embedded document in each collection; employee collection contains children as an embedded document and department collection has project as an embedded document; in this structure and two reference documents in employee collection one for the department and the other one for the projects.

All the queries execution in this structure used about 2 parts of the second. Query 3 took more time as it retrieves data elated to an embedded document from the two collections.

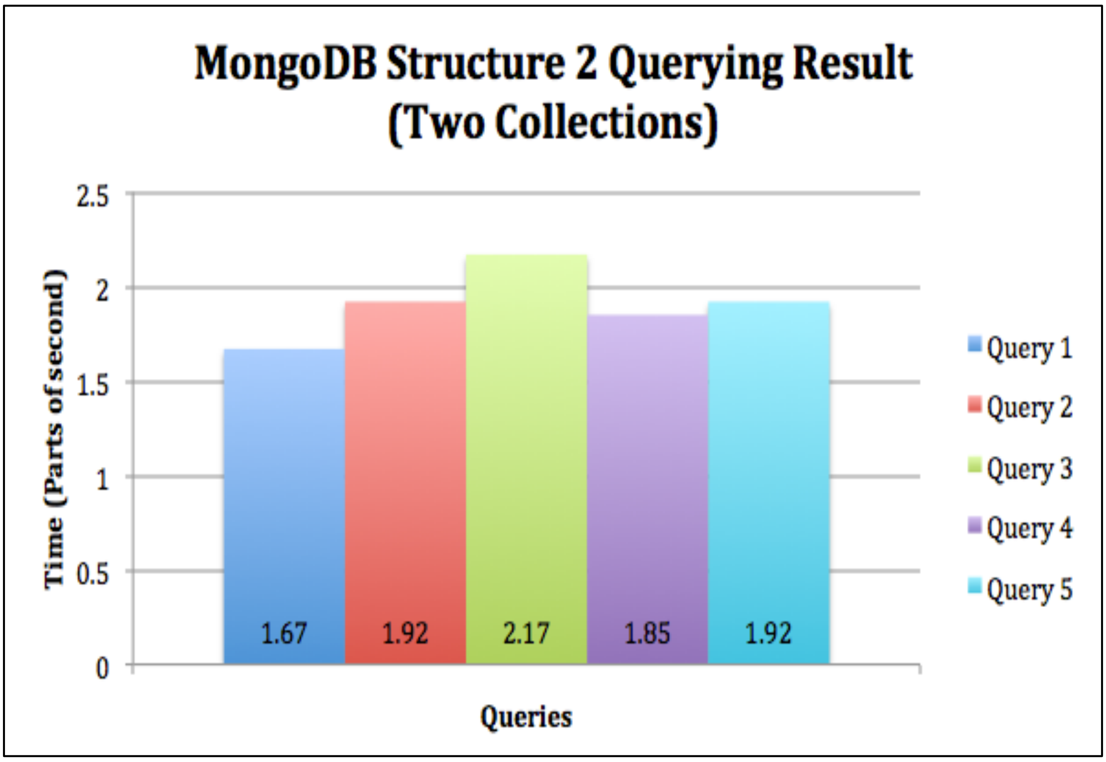


Figure 10: MongoDB Structure 2 - query results

3.1.4 Results of queries on Mongo database (Structure 3)

The chart in Figure 11 below shows the time consumed in executing each one of the 5 queries used in the research when data is presented in five Mongo collection with reference documents. Each document has a reference relationship with one or more documents in the database.

It is clear from the chart that the execution time for the queries vary based on which documents are used to find the requested data. The time needed to get data in query 1 is very long, as it is required to access all the documents using the reference variable involved in the query. Query 2 as well is quite long as it uses the reference variables and many documents. Query 5 is using lookup function and the time is not very short compared to the time of query 3 and 4. The fastest query to be executed is query 3 because the query is collecting data from 3 collections only using the reference keys employee id, project id and department id only.

R. Lawrence

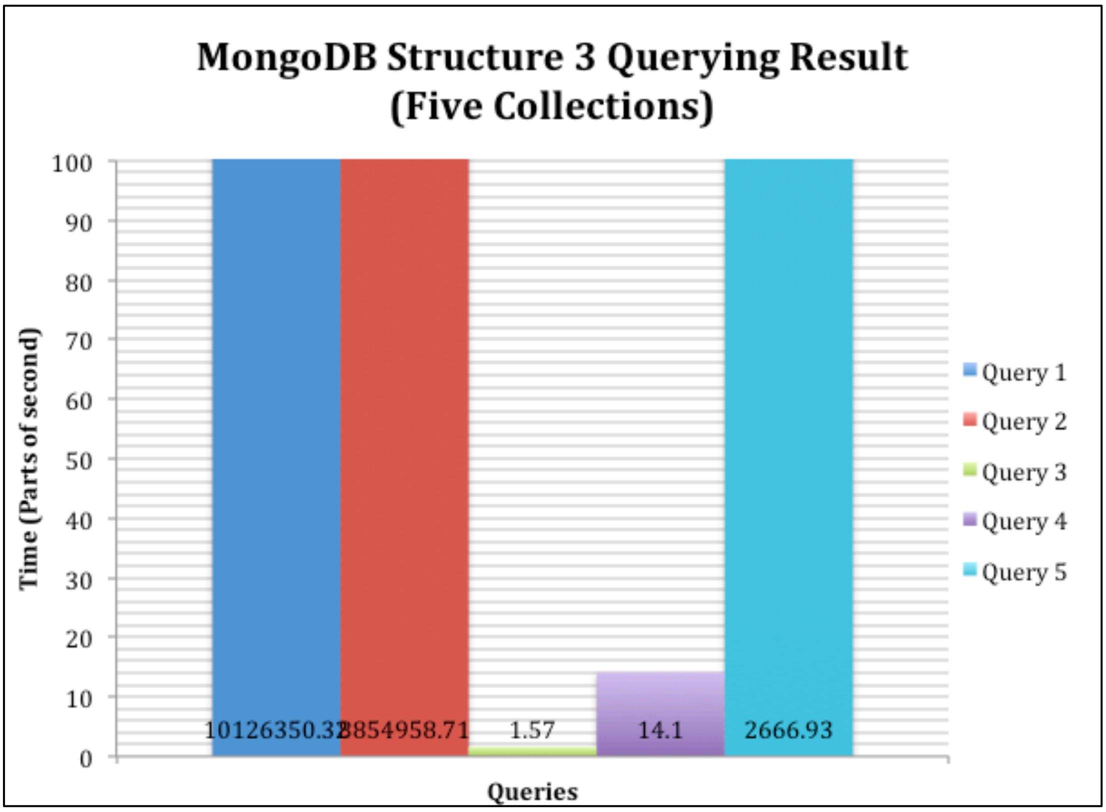


Figure 11: MongoDB Structure 3 - query results

3.2 Data Query Results based on Query

In this section we will display the results of executing the same query on all the database structure. The aim of doing this is to know which structure performs better in terms of executing time.

3.2.1 Results of executing Query 1

The chart in Figure 12 below shows that MySQL database achieved the best time when the query needs all the information about one employee as primary and foreign keys are used to collect the data. The time is for structure 2 (two collections with both embedded and reference documents) because there is no need to access second level embedded document. For MonogDB structure 1 (one collection with embedded documents), the time consumed is very close to structure 2. Regarding MongoDB structure 3 the chart shows that executing this query will need about 101263.5 seconds, which means about 28 hours because of using the lookup functions in all the data retrieval requests as explained before.

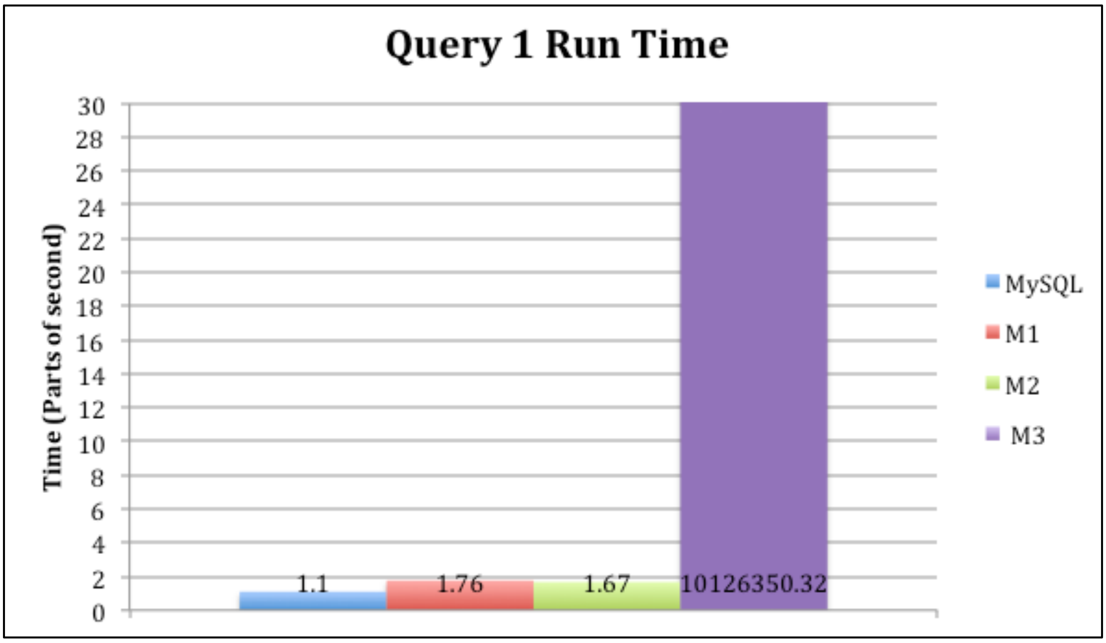


Figure 12: Query 1 run time results

3.2.2 Results of executing Query 2

The best time achieved in executing query 2 is with MongoDB structure 2; it was about 1.92 parts of the second as shown in the chart in Figure 13. MonogDB structure 1 execution time for query 2 was very close to MongoDB structure 2 execution time. A noticeable increase in the execution time of the same query in MySQL structure, as the query will access projects and works_on tables to find employee id then will get all the information related to that employee. All of the three previous structures execution time was very short in compare with the execution time of MongoDB structure 3 that reached about 24 hours to get the results. The reason behind the very high execution time in structures 3 is the reference relationship between the collections and the need to access all the collections after finding the employeeID required.

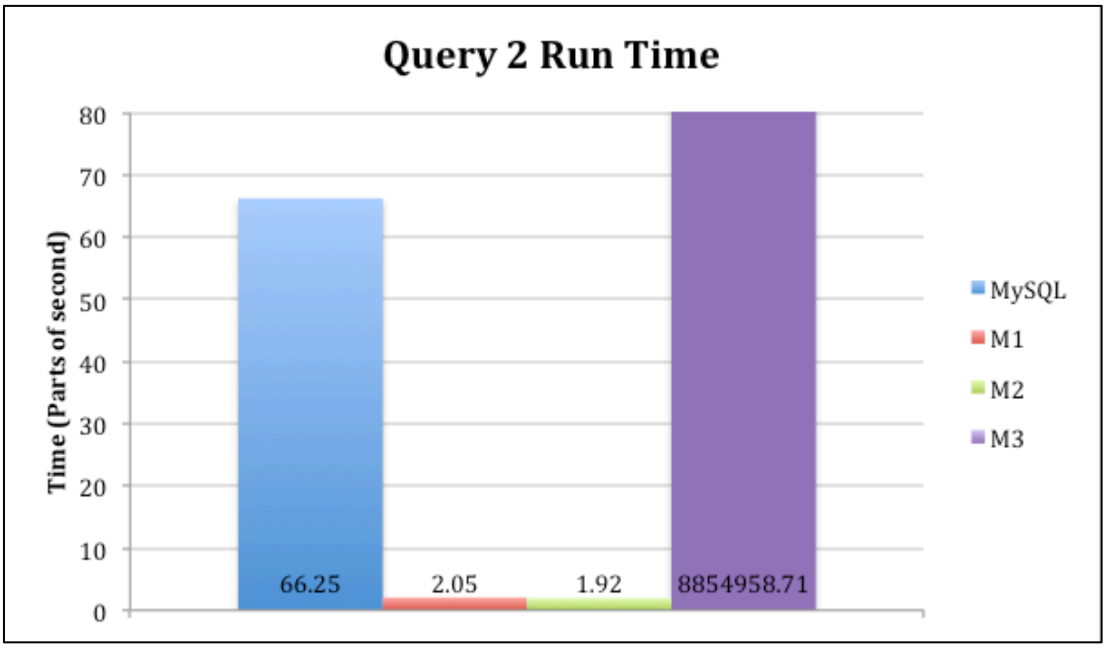


Figure 13: Query 2 run time results

3.2.3 Results of executing Query 3

MySQL database structure recorded the worst time in executing query 3. The time needed to retrieve data was about 65.95 parts of second refer to Table 3 for the query description, the process will access the works_on table then the project table and will use the id_employee, id_department, and id_project to collect all the required data and this is no happening in the other structures as the design is different. While MongoDB structures1, 2 and 3 results were close to each other as what is displayed in Figure 14. We can conclude that any representation in MongoDB is better that MySQL representation of data in executing this kind of queries that does not rely on key data.

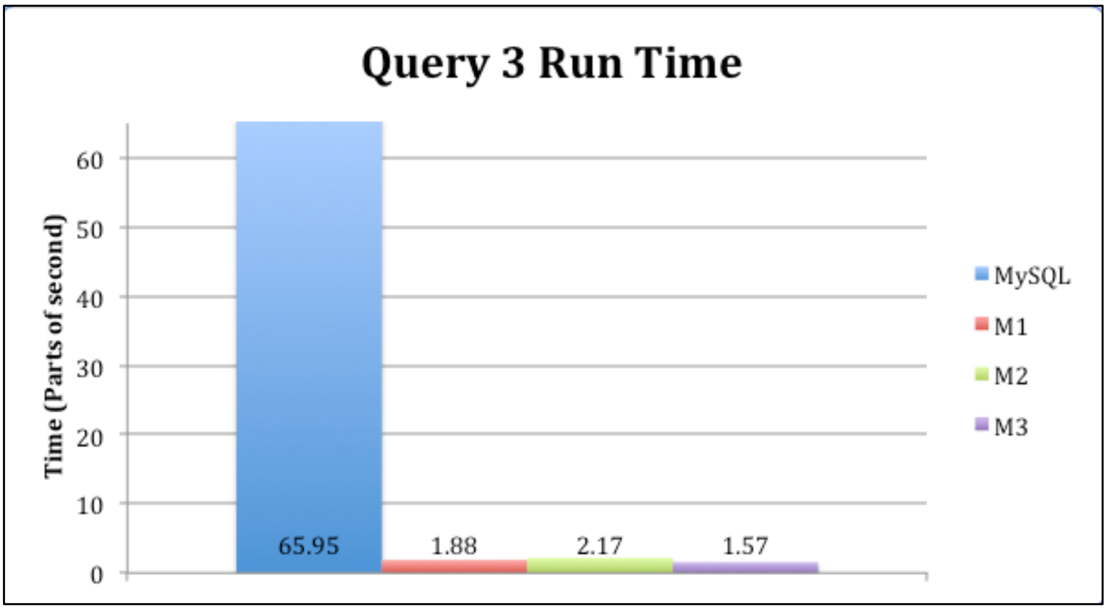


Figure 14: Query 3 run time results

3.2.4 Results of executing Query 4

The chart in Figure 15 shows the execution time of query 4 in the 4 different database structures used in the research. It is obvious that MongoDB structure 1 is not a good choice to execute this type of queries. MongoDB structure 2 is the best structure for retrieving data according the query 4 requirements. The time needed in

MongoDB structure 3 was about 14.10 parts of the second, and MySQL structure used almost 69 parts of the second to complete the same task as it collects data about the project using project id and department id and then retrieves the department information from the department table.

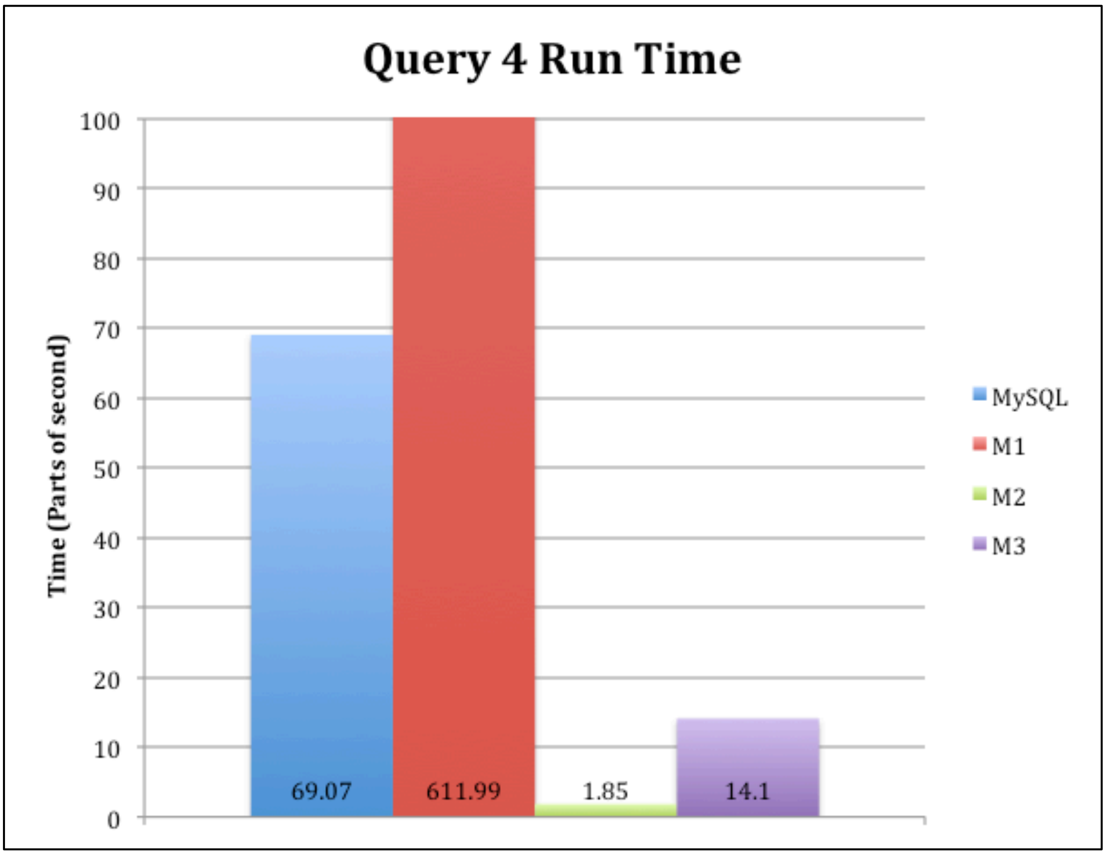


Figure 15: Query 4 run time results

3.2.5 Results of executing Query 5

It is very clear in the chart represented in Figure 16 that MongoDB structures 1 and 2 completed the task in much less time than MongoDB structure 3 and MySQL structure. The time difference is very clear between the two groups. Even though MongoDB structure 1 is better than structure 2 but both of them have close results.

While MySQL needed time to execute the query was very less compared to MongoDB structure 3 but both of then used long time for this query.

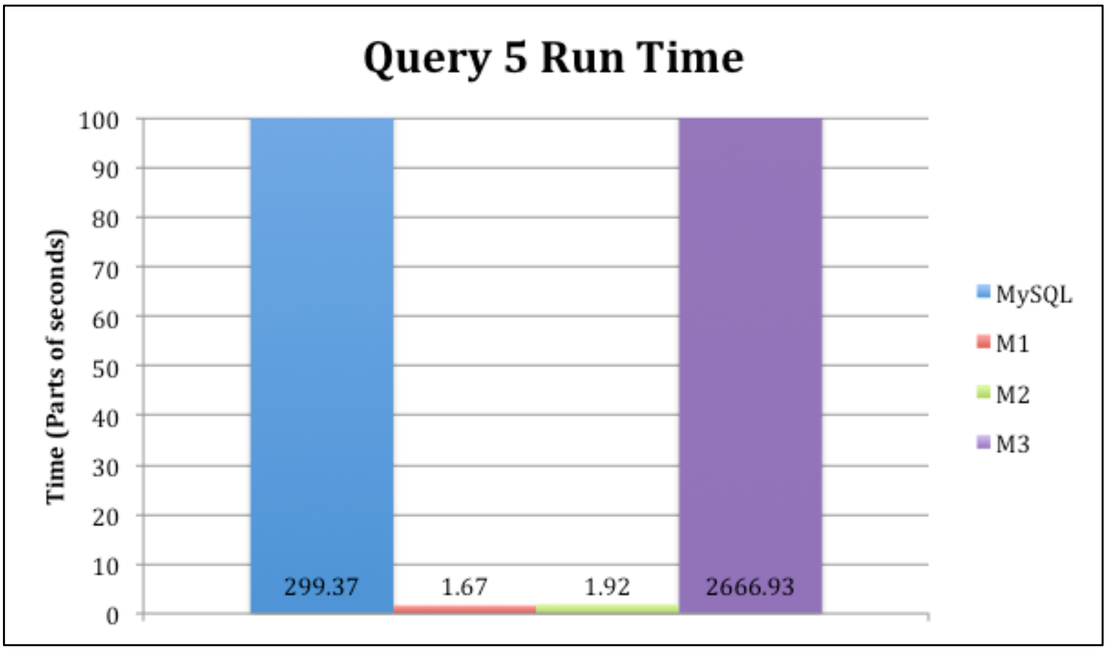


Figure 16: Query 5 run time results

3.3 Final Findings

After discussing the results in the previous two sections 3.1 and 3.2 and explaining in details each one of the data structures and how they act with different types of queries. The chart in Figure 17 represents the average of the execution time for all the structure. In comparison with MySQL database structure, the green bar that represents the MongoDB structure 2 shows stable and much better execution time for all the types of the suggested queries than MySQL results. MongoDB structure 2 represents data in two collections with both one level embedded document and reference documents. MongoDB structure 1 representation shows better results than MySQL as well except for query 4 as it requires collecting data from the embedded documents and this problem can be solved when using indexing in embedded documents [6]. The last MongoDB structure, structure 3, requires longer time than MySQL to execute the queries except query 3 and 4, but even though the very long execution time for query 1 and 2 make it inconvenient to use this representation instead of MySQL.

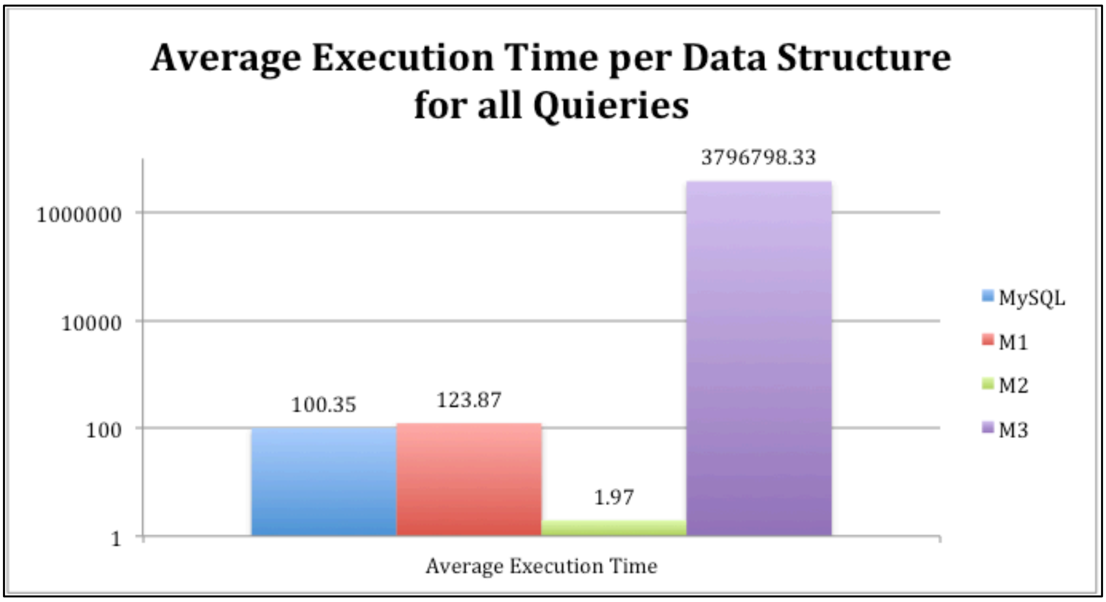


Figure 17: Summary of final results

As my research main objective is to find out which is the best way to represent a complete SQL database in NoSQL. We have chosen MongoDB as NoSQL database to represent MySQL database that contains 5 tables with different relationships and millions of records.

The research final output recommends any company that is willing to move from SQL to NoSQL and has a big number of records stored in tables with different types of relationships to represent data using MongoDB collections that include one level embedded documents and reference relationships between the collections, as this representation proved the most efficient time in executing different types and levels of queries.

Chapter 4: Conclusion and Future Work

Data growth is one of the most significant issues nowadays. As a response to this growth enterprises are moving towards using NoSQL databases instead of the existing SQL database [20]. The main idea of this research came to find the best way to represent the current SQL database in an enterprise with NoSQL database specifically MongoDB.

4.1 Conclusion

In conclusion, a database that contains five tables with a variety of relationships many-to-one, and many-to-many was used. Also the huge amount of data stored in all the structures about 2 millions record/document. The research compares clearly between the performance of retrieving data from different MongDB representation of data and the result shows that in some cases using more than one collection to represent huge data with complex relationships is better than keeping all the data in one document.

After filling all the tables in the MySQL database with random data, the five queries were executed 20 times and the average time was recorded for each query. Then 3 different structures of MongoDB were designed to include all the different relationships in MongoDB. The first structure has one collection with two levels of embedding documents. The second one contains two collections with one level of embedding document in both and reference relationship between them as well. The third structure has five collections with only reference relationships between them. After designing the new structures, they were filled with the exact data saved in

MySQL database to have fair results. And then the five queries were executed 20 times and the average time was recorded for each query.

In the research the results from three different perspectives were discussed as follows: Firstly, compare all the result based on the structure type. Secondly, compare the result of each query with the registered results of the other structures for the same query. Lastly, comparing the average execution time of the queries results for all the structures to find out that the best structure to implement when the enterprise decides to move to NoSQL.

The research findings indicates that using two collections with one level of embedding documents and reference relationship between the collections to represent the current MySQL database because the execution time recorded in all the queries was the least with no odd readings.

4.2 Future Work and Open Issues

In this research, the default indexing in both MongoDB and MySQL database were used. The results in this research may change when using indexes in both databases MongoDB and MySQL. Trying to include more tables in the database then testing who it will affect the results will enrich the findings of this research. Those two points can be a starting point for a new research that will help in deciding the best NoSQL representation of SQL existing model.

One the important limitation that affected this research is maximum document size in MongoDB. The maximum document size in MongoDB is 16 Megabytes [22]. This limitation will restrict the design of the mongo collection. For example we failed to use the department as the main document and add all the other data as embedded documents within this document due to this limitation of MongoDB.

References

[1] T. Jia, X. Zhao, Z. Wang, D. Gong and G. Ding, "Model Transformation and Data Migration from Relational Database to MongoDB," 2016 IEEE International Congress on Big Data (BigData Congress), San Francisco, CA, 2016, pp. 60-67.doi: 10.1109/BigDataCongress.2016.16

[2] G. Zhao, Q. Lin, L. Li and Z. Li, "Schema Conversion Model of SQL Database to NoSQL," 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Guangdong, 2014, pp. 355-362. doi: 10.1109/3PGCIC.2014.137

[3] G. Zhao, L. Li, Z. Li and Q. Lin, "Multiple Nested Schema of HBase for Migration from SQL," 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Guangdong, 2014, pp. 338-343. doi: 10.1109/3PGCIC.2014.127

[4] Z. Wei-ping, L. Ming-xin and C. Huan, "Using MongoDB to implement textbook management system instead of MySQL," 2011 IEEE 3rd International Conference on Communication Software and Networks, Xi'an, 2011, pp. 303-305. doi: 10.1109/ICCSN.2011.6013720

[5] A. Boicea, F. Radulescu and L. I. Agapin, "MongoDB vs Oracle -- Database Comparison," 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, Bucharest, 2012, pp. 330-335. doi: 10.1109/EIDWT.2012.32

[6] J. S. van der Veen, B. van der Waaij and R. J. Meijer, "Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual," 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, 2012, pp. 431-438. doi: 10.1109/CLOUD.2012.18

[7] G. Zhao, W. Huang, S. Liang and Y. Tang, "Modeling MongoDB with Relational Model," 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, Xi'an, 2013, pp. 115-121. doi: 10.1109/EIDWT.2013.25

[8] Lv, Q., & Xie, W, “A Real-Time Log Analyzer Based on MongoDB. In Applied Mechanics and Materials” , Trans Tech Publications , Vol. 571, pp. 497-501, 2014.

[9] R. Lawrence, "Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB," 2014 International Conference on Computational Science and Computational Intelligence, Las Vegas, NV, 2014, pp. 285-290. doi: 10.1109/CSCI.2014.56

[10] C. Győrödi, R. Győrödi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL," 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, 2015, pp. 1-6. doi: 10.1109/EMES.2015.7158433

[11] V. Guimarães et al., "A study of genomic data provenance in NoSQL document-oriented database systems," 2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Washington, DC, 2015, pp. 1525-1531. doi: 10.1109/BIBM.2015.7359902

[12] M. G. Jung, S. A. Youn, J. Bae and Y. L. Choi, "A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment," 2015 8th International Conference on Database Theory and Application (DTA), Jeju, 2015, pp. 14-17. doi: 10.1109/DTA.2015.14

[13] Y. Gu, S. Shen, J. Wang and J. U. Kim, "Application of NoSQL database MongoDB," 2015 IEEE International Conference on Consumer Electronics - Taiwan, Taipei, 2015, pp. 158-159. doi: 10.1109/ICCE-TW.2015.7216831

[14] S. Chickerur, A. Goudar and A. Kinnerkar, "Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications," 2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA), Jeju, 2015, pp. 41-47. doi: 10.1109/ASEA.2015.19

[15] S. H. Aboutorabi, M. Rezapour, M. Moradi and N. Ghadiri, "Performance evaluation of SQL and MongoDB databases for big e-commerce data," 2015 International Symposium on Computer Science and Software Engineering (CSSE), Tabriz, 2015, pp. 1-7. doi: 10.1109/CSICSSE.2015.7369245

[16] C. H. Lee and Y. L. Zheng, "SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems," 2015 IEEE International Conference on Systems, Man, and Cybernetics, Kowloon, 2015, pp. 2022-2026. doi: 10.1109/SMC.2015.353

[17] L. Stanescu, M. Brezovan and D. D. Burdescu, "Automatic mapping of MySQL databases to NoSQL MongoDB," 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), Gdansk, 2016, pp. 837-840.

[18] A. Goyal, A. Swaminathan, R. Pande and V. Attar, "Cross platform (RDBMS to NoSQL) database validation tool using bloom filter," 2016 International Conference on Recent Trends in Information Technology (ICRTIT), Chennai, 2016, pp. 1-5. doi: 10.1109/ICRTIT.2016.7569537

[19] P. Gómez, R. Casallas and C. Roncancio, "Data schema does matter, even in NoSQL systems!," 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS), Grenoble, 2016, pp. 1-6. doi:

10.1109/RCIS.2016.7549340

[20] Liao, Y. T., Zhou, J., Lu, C. H., Chen, S. C., Hsu, C. H., Chen, W., ... & Chung, Y. C., "Data adapter for querying and transformation between SQL and NoSQL database," 2016 Future Generation Computer Systems, pp. 111-121.

[21] V. Anand and C. M. Rao, "MongoDB and Oracle NoSQL: A technical critique for design decisions," 2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS), Pudukkottai, 2016, pp. 1-4. doi: 10.1109/ICETETS.2016.7602984

[22] "MongoDB Limits and Thresholds". (n.d.). Retrieved September 27, 2017, from https://docs.MongoDB.com/manual/reference/limits/

[23] Prof. Dott-Ing. Roberto V. Zicari: "Introduction to ODBMS". Retrieved September 27, 2017, from http://www.odbms.org/introduction-to-odbms/definition/

[24] J.J. Simmins, P.T. Myrda & B.Taube, "Advanced utility analytics with object oriented database technology," T&D Conference and Exposition, 2014 IEEE PES, Chicago, IL, USA, 2014, pp. 1-5. doi: 10.1109/TDC.2014.6863346