A NEW AUDITING MECHANISM FOR OPEN SOURCE NOSQL DATABASE – A CASE STUDY ON OPEN SOURCE MONGODB DATABASE

HANY HEIDAR HUSSEIN MOHAMED

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

SCHOOL OF COMPUTING

COLLEGE OF ARTS AND SCIENCES

UNIVERSITI UTARA MALAYSIA

2015

Permission to Use

Abstrak

MongoDB adalah satu contoh sistem pengurusan pangkalan data NoSQL yang agak baru di pasaran pangkalan data dan ia digunakan dalam banyak projek penting dan produk. Analisis Keselamatan untuk MongoDB mendedahkan bahawa ia tidak memberikan apa-apa kemudahan untuk tindakan audit dilakukan dalam pangkalan data. Baru-baru ini, syarikat MongoDB cuba untuk membetulkan jurang pengauditan dengan menyediakan MongoDB perusahaan baru versi 2.6 (8 April 2014). Sistem pengauditan boleh merakam operasi berikut: skema (DDL), set replika, pengesahan dan kebenaran, dan operasi umum. Tetapi malangnya ia masih tidak boleh merakam Data Manipulasi Bahasa (DML). Oleh itu, kajian ini bertujuan untuk meningkatkan fungsi pengauditan di MongoDB dengan membentangkan satu mekanisme baru bagi pengauditan pangkalan data NoSQL MongoDB untuk memasukkan Data Manipulasi Bahasa (DML) / CRUD (Membuat, Baca, Kemaskini dan memadam) operasi.

Kata Kunci: data big, NoSQL, MongoDB, MongoDB pengauditan

Abstract

MongoDB as a NoSQL database management system is relatively new on the database market and it is used in many important projects and products. Security analysis for MongoDB revealed that it doesn't provide any facilities for auditing actions performed in the database. Recently, MongoDB company tried to rectify the auditing gap by providing MongoDB new enterprise version 2.6 (8th of April 2014). The auditing system logs operations information including; schema data definition language operations and operations related to replica set in addition to operations of authentication and authorization, and eventually general operations. But unfortunately still cannot record Data Manipulation Language (DML). Thus, this study aims to improve the auditing functionality in MongoDB by presenting a new mechanism for auditing NoSQL MongoDB database to include Data Manipulation Language (DML)/ CRUD (Create, Read, Update and delete) operations.

Keywords: Big data, NoSQL, MongoDB, MongoDB auditing

Acknowledgement

All praises and thanks due to Almighty Allah, the most gracious and the most merciful for lightening my way throughout the completion of this valuable thesis. I adore His benevolence and mercy, without his kindness, I will not be able to complete this study especially as I was thousand miles away from my beloved country (Egypt). Also I would like to thanks to my Supervisor Dr. Massudi Mahmuddin. Without his patient support, enlightened guidance, it is impossible for me to complete and enhance the quality of my work.

My deepest and heartfelt gratitude, loves, thanks and appreciation for my dearest parents and my beloved siblings who are a part of my happiness, success, and the inspiration that led me for the quest for knowledge and self-empowerment through night and day. I hope I can put a smile on their faces for giving back their remendous support and encouragement, patience, unconditional love, and prayers for me. Thank you for giving me the strength to chase and reach my dreams.

Thank You All.

"This Thesis is only the beginning of my journey."

HANY HEIDAR HUSSEIN MOHAMED

UUM University, Kedah, Malaysia

Monday, January 12, 2015

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
BSON	Binary JavaScript Object Notation
CRUD	Create Read Update Delete
DDL	Data Definition Language
DML	Data Manipulation Language
DBA	Database Administrators
JSON	JavaScript Object Notation
NoSQL	Not Only SQL
RBAC	Role Based Access Control
RDMS	Relational Database Management System
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
OS	Operating System

SQL	Structured Query Language
TCP/IP	Transmission Control Protocol /Internet Protocol

CHAPTER ONE
INTRODUCTION

1.1 Background

The term NoSQL is used first time by Mr. Carlo Strozzi (1998) to name his lightweight open source relational database. The system did not expose the standard SQL (Structure Query Language) interface. There is a series of database following NoSQL (Not Only SQL) standards. The term “Not Only SQL” is also used for these databases that provide storage and retrieval mechanism with less constrained consistency models than traditional relational databases (Mohamed, Altrafi, & Ismail, 2014).

The last three decades were ruled by the traditional relational database management systems such as DB2, MS SQL Server and Oracle (Bonnet, Laurent, Sala, Laurent, & Sicard, 2011). They have the standard SQL. Due to the growing web scale applications such as Facebook, mobile applications and RFID (Radio Frequency Identification) the Internet has become an essential part of the world today.

Everyday zettabytes of data are being generated due to these applications. Due to changing need of applications and databases, the traditional relational databases are proved to be weak in distributed environment. This made NoSQL databases to get importance and preference. Being schema free, elastic and scalable, NoSQL databases appeared to be effective(Kanade, Gopal, & Kanade, 2013).

Simple design, horizontal scalability and availability NoSQL databases have led to gain the popularity worldwide. Recently they are widely used in big data and real time web applications like Facebook, Yahoo, Google, and Amazon. With the advent of the Web 2.0 NoSQL has entered the database market (Kanade et al., 2014).

Mainly there are five categories of NoSQL databases like Key-Value Store, Document-Store, Column-Oriented, Graph Database, and data structure store. Cassandra, MongoDB, BigTable, neo4j and Redis are the examples of these databases respectively. The databases like XML and Object Oriented Databases come under the NoSQL category.

With the venture funding and open-source movement provided by the smaller IT companies databases like Couchbase, MongoDB and Riak have come up. Oracle also has enhanced its old Berkeley DB with some NoSQL features which is called as Oracle NoSQL. A product HBase by IBM has been targeted by the big data now (Kanade et al., 2013).

Many organizations used NoSQL databases, however studies on NoSQL databases security features found that auditing in addition to other security aspects are overlooked. Auditing for NoSQL implementations as database management systems is very important for an enterprise to protect its sensitive information. Auditing as database security feature is needed as a shield against a potential risk from trusted users who own or can obtain the correct credentials.

If an enterprise or an organization that uses NoSQL database products has not implemented such measures as database monitoring or auditing, then their invaluable assets reside in database are at high risk. Meanwhile, auditing helps enterprises in forensic analysis in addition, to protect sensitive information. The only solution to this problem is to implement database auditing. Database auditing involves observing a database so as to be aware of the actions of database users (Liu & Huang, 2009).

One of the leading and widespread of NoSQL databases is MongoDB. MongoDB is an open-source document database, and the leading NoSQL database developed by the software company 10gen (now MongoDB Inc.) in October 2007 and written in C++. The objects are stored serialized as BSON (Binary JavaScript Object Notation). The objects do not share the similar structure or same fields as well the similar fields do not need to have the same type, thus allowing a flexible schema storage (Li & Manoharan, 2013).

1.2 Problem statement

MongoDB as a NoSQL database management system is relatively new on the database market and it is used in many important projects and products, such as: MTV Networks, Craiglist, Disney Interactive Media Group, Sourceforge, Wordnik, Yabblr, SecondMarket, The Guardian, Forbes, The New York Times, bit.ly, GitHub, FourSquare, Disqus and PiCloud (Boicea, Radulescu, & Agapin, 2012).

In MongoDB, there are no database schemas or tables. MongoDB instead uses a "collection" which is similar to a table and "documents" which are similar to rows, to store the data and schema information. MongoDB automatically generates a primary key (id) to uniquely identify each document. The id and document are conceptually similar to a key-value pair (need other sentence for coherence).

Okman, Gonen and Abramoy (2011) analysed MongoDB Security features and they found that MongoDB doesn't provide any facilities for auditing actions performed in the database. When a new namespace (database) is created, Mongo will write a line in the log about data file creation, but after the data files are allocated, nothing new appears in the log for any subsequent insertions, updates or queries.

Grolinger, Higashino, Tiwari and Capretz (2013) identified challenges and opportunities in NoSQL databases. Referring to MongoDB auditing, they stated that MongoDB doesn't present auditing functionalities for events on database. Moreover, they referred to that auditing functionalities are usually related to the creation of an audit trail that logs records of events that occurred in a data stores. This is especially important in forensic analysis of security events.

Recently, MongoDB company tried to rectify the auditing gap by providing MongoDB new enterprise version 2.6 (8th of April 2014). The auditing system logs operations information including; schema data definition language operations and operations related to replica set in addition to operations of authentication and authorization, and eventually general operations. But unfortunately still cannot record Data Manipulation Language (DML) (MongoDB, 2014).

Most organizations require that all user-based Data Manipulation Language (DML) and/or CRUD (Create, Read, Update and Delete) operations performed against production databases be logged (MongoDB, 2014). This request extends the MongoDB auditing framework, introduced in version 2.6, to include logging of all user queries and DML/CRUD operations including:

- query/read - any operation that returns data.
- insert – any operation that adds data to a database.
- update – any operation that changes data on a database.
- delete – any operation that removes data from a database.

Upon the above studies and literature review, it is clear that there is a need to enhance the auditing functionality in MongoDB.

1.3 Research Questions

There are two main questions for this study as follows:

1. What are MongoDB auditing features?
2. How to improve auditing in MongoDB?

1.4 Research Objectives

Thus, our research aims to improve auditing in MongoDB to include Data Manipulation Language (DML)/ CRUD (Create, Read, Update and delete) operations. This study follows the research objectives given below:

1. To identify MongoDB auditing features.
2. To improve auditing in MongoDB by developing a new auditing mechanism.
3. To evaluate the proposed mechanism.

1.5 Research Scope

The research concentrates on NoSQL database document data model. The study is limited to MongoDB as document oriented data model implementation. Precisely, this research focuses on auditing DML/CRUD operations in MongoDB.

1.6 Contributions

1. Identifying auditing features that assisting MongoDB users and vendors to enhance what already exist beside develop new other ones.
2. Improving auditing in MongoDB that is helping in monitoring and protecting the sensitive data stored in MongoDB database.

1.7 Organization of Report

This study is organized in five chapters. An outline of the essential contents of the following chapters is expressed as follow:

Chapter 2: reviews related studies on NoSQL Database and MongoDB.

Chapter 3: discusses the research methodology that was used to construct an auditing mechanism.

Chapter 4: presents the results and evaluation.

Chapter 5: concludes the findings of the study.

CHAPTER TWO

LITERATURE REVIEW

2.1 NoSQL Database

2.1.1 **Overview**

NoSQL stands for Not Only SQL and also means No Relational or No RDBMS. NoSQL is a term for all data stores that do not follow the traditional RDBMS rules (Pozzani, 2013). NoSQL databases are different from the relational databases by not possessing a fixed predefined schema to follow by data architects and engineers. But, NoSQL data models include, key-value, graph based, column and document oriented. Various architectures are qualified for different requirements of data management (Narde, 2013).

The architecture of NoSQL database uses a cluster of servers. Most of the servers in the cluster play the role of data nodes, the node which maintains data sets. And there are few nodes in the cluster which plays role of monitoring and balancing the cluster, these nodes are called in different names in different databases. In HBase these nodes are called zookeepers, in MongoDB those are called config servers. And there will be metadata node which plays the role of master node assigning data partition/shards to data nodes or acts as a router to the requests.

2.1.2 NoSQL Data Models

The NoSQL database consists of several data models. Various distinct interpretations have been proposed for NoSQL data models that have guided to various sub-classifications (Tudorica, & Bucur, 2011). Furthermore, these sub- classifications divide the different NoSQL data stores into four principal data models: key-value data model, Column data model, document data model, and graph data model (Hecht & Jablonski, 2011).

Table 2.1

List of companies using NoSQL database with its categories

Organization	**NoSQL Database**	**Data Model**
Adobe	HBase	Column
Amazon	Dynamo – SimpleDB	Key-value - Document
Facebook	Cassandra – Neo4j	Column - Graph
eBay	MongoDB - Cassandra	Document - Column
Linkedin	Voldmort	Key-value
Twitter	Cassandra	Column
Google	BigTable	Column

Table 2.1 shows a list of International companies using NoSQL databases upon its categories.

Table 2.2

NoSQL data stores (Grolinger, Higashino,Tiwari & Capretz, 2013)

NoSQL Data Stores	**Name & Source**	**Querying**	**Licence**
Key-value stores	**Redis** http://redis.io	Does not provide SQL-like querying	Open source: BSD (Berkeley Software Distribution)
	Memcached http://memcached.org	Does not provide SQL-like querying	Open source: BSD 3-clause License
	BerkeleyDB http://www.oracle.com/us/products/database/berkeleydb/ overview/index.html	SQLite	Closed source: Oracle Sleepycat license
	Voldemort http://www.projectvoldemort. com/voldemort	No	Open source: Apache 2.0 License
	Riak http://basho.com/riak	Riak search, secondary indices	Open source: Apache 2.0 License

Column family stores	**Cassandra** http://cassandra.apache.org	Cassandra query language	Open source: Apache 2.0 License
	HBase http://hbase.apache.org	No, could be used with Hive	Open source: Apache 2.0 License
	DynamoDB (Amazon service) http://aws.amazon.com/dynamodb	Proprietary	Closed source: Pricing as pay per-use basis
	Amazon SimpleDB (Amazon service) http://aws.amazon.com/simpledb	Amazon proprietary	Closed source: Pricing as pay per-use basis
Document stores	**MongoDB** http://www.mongodb.org	Proprietary	Open source: Free GNU AGPL v3.0 license
	CouchDB http://couchdb.apache.org	SQL like UnQL, under development	Open source: Apache 2.0 License
	Couchbase Server http://www.couchbase.com	No	Open source: Free Community Edition. Paid Enterprise Edition

Graph databases	**Neo4J** http://www.neo4j.org	Cypher, Gremlin and SparQL	Open source license: NTCL + (A)GPLv3
	HyperGraphDB www.hypergraphdb.org/	SQL like querying	Open source license: GNU LGPLv3
	Allegro Graph http://www.franz.com/agraph/allegrograph	SparQL and Prolog	Closed source: free, developer and enterprise versions

2.1.2.1 Key-value Data Model

Key-value data model consists of key-value pairs that like a dictionary or an associative map (Hecht & Jablonski, 2011). One of drawbacks for this data model that it is not convenient for structures or relations cases. Therefore, if there is a need to this functionality it should be performed on application level (Grolinger, Higashino,Tiwari & Capretz, 2013).

2.1.2.2 Column Data Model

Most of column oriented stores are based on Bigtable of Google (Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, & Gruber, 2008). The data in Bigtable are saved in a column as a data model. Furthermore, the dataset of Bigtable has a row key also named as a primary key for each set of rows. Each row is consists of a group of columns, and various rows can have diverse columns.

Identically to key-value stores, the row key is similar to the key in key-value data model and further the set of column is like the of the row key. But, each column used as a key for a column or more within. Meanwhile each column includes a name-value pair and also introduced what is called super-columns through grouping various columns.

Generally, indexing and querying are more powerful in Column stores data model. As same as previously mention about key-value data model relating to structures and relations, also here with column data model any requesting relations should be done in the application (Grolinger, Higashino,Tiwari & Capretz, 2013).

2.1.2.3 Document Data Model

Document stores use keys to find documents within the data store. Mostly document data model includes documents that are written with JSON (JavaScript Object Notation) or other related format. For instance, both of CouchDB use the format of JSON for data storage, however MongoDB stores data in Binary JSON. Document data model are convenient in cases that application's input data can be stored in a document-oriented form.

A document has various data structures and cannot follow a pre-defined constant schema. Additionally, MongoDB provides the ability of collecting the documents into what is called collections. Furthermore, in every collection, a document and each document should include a unique key. Indexing and querying functionalities are supported in document stores data model.

2.1.2.4 Graph Data Model

Graph data model is built on the basis of graph theory. A graph as a mathematical terminology used to define a group of objects, named as nodes or vertices, and interconnect between them. Graph database model can store the relationships between various data nodes. Graph data model is designed for dealing with interconnected data and crossing relationships between various entities. They are convenient in many cases including social networking, dependency analysis, recommendation system and, pattern recognition. Certain graph databases like Neo4J are completely ACID (Atomicity, Consistency, Isolation, Durability) compliant (Hecht & Jablonski, 2011; Buerli, & Obispo, 2012).

2.2 Importance of NoSQL in Big Data Applications

Recently, numerous NoSQL ("Not Only SQL") systems have been released and widely adopted in many domains. NoSQL systems have been developed to support applications not well served by relational systems, often involving Big Data processing (Lawrence, 2014).

NoSQL systems can be categorized as key-value stores, document stores, and graph databases. Importantly, there are no common standard APIs for accessing the different NoSQL systems or standard query languages such as SQL. This reduces portability and requires system-specific code. Although most NoSQL systems do not support SQL, there is no fundamental reason why they could not. The "NoSQL" label is a misnomer. The value of these systems has nothing to do with SQL support, but rather on their different architectural design decisions in order to achieve scalability and performance (Lawrence, 2014).

SQL is valuable as it is a standard that allows portability, expressiveness, and has a massive installed base of trained users. As NoSQL systems evolve, there has been recognition of the value of SQL, and several commercial systems are hybrids combining a SQL relational processor with a Big Data (MapReduce) query processor (Lawrence, 2014).

2.3 NoSQL Database Security Issues

In this section security built-into the NoSQL databases environment is reviewed which are entrusted in handling Big Data, further measuring the drawbacks of these systems. Likewise, security gap inherent in the NoSQL database system environments is revealed and exploring best methods to secure these systems. The volume, variety and ratio of generated data for processing and storage results in huge amounts of data that need to be safely and correctly secured and saved.

A survey of the top big data vendors as well as deployments for services extending from server and storage hardware, database software, analytics applications and other associated services have revealed that vast amounts of valuable and sensitive data are being handled through the various applications across many platforms all over the world generated by humans or by machines that routinely access and use Web and mobile applications. These data, owned by its organizations is highly worthy, beneficial, and a matter of and follows privacy laws and compliance regulations which have to be protected.

2.3.1 Threats Posed By Distributed Environments

NoSQL database environment related Nodes are distributed subsequently resulting in vast simultaneous processing. Accordingly, this environment is ready and vulnerable for attacks and threats that is very difficult to secure particularly in case of NoSQL database system across multiple distributed nodes. In addition, deciding the place from where you access database system, which may be at the Clients side locations or at another remote location which raises the potential of unauthorized access (Kadebu, Mapanga, 2014).

2.3.2 Safeguarding Integrity

The process of protecting and securing integrity issues is more difficult and complex in NoSQL database system due to its nature as heterogeneous in comparing with homogeneous systems. Moreover, there is no central control and its schema-less nature makes it much harder to impose integrity restrictions (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.3 Communication between Nodes

All communication protocols such as nodes of data connect to what is called Name Nodes and all are layered on top of the TCP/IP mainly relying on RPC over TCP/IP. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol in the NoSQL distributed environment. NoSQL database Systems with RPC ports exposed to the Distributed environment are especially vulnerable. Security concerns emanate as nodes interact through message passing, because communication is not secure (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.4 Sharded Data/Fragmented Data

NoSQL databases horizontally divides data into slices, shares and combines them over several servers. Data from a variety of nodes move to and from in the NoSQL database environment which is distributed across multiple servers. Furthermore, here an example is that at one organisation which has clusters with up to 4000 nodes, and about 65 million files and 80 million blocks.

Accordingly, this data automatically moves to different locations for large inter/intra-clustering using parallel copying mechanism of MapReduce in order to copy parts of the source data into file system as a destination. The process of maintaining replicated shards of data which combines security passwords is complicated and needs expensive computation. Besides, this process is more vulnerable to failure and raises the risk of theft. This model causes difficulties in protecting data as it becomes replicated and moves in different locations as needed since it is not centralised (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.5 Compromised Clients

Clients connecting to NoSQL databases are able to see and access resource managers and several nodes in direct. In cases that combines malicious data which has been propagated from a compromised location, the whole system is compromised. Securing nodes, name servers and those clients is considered difficult particularly when there is no central management security point (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.6 Protection of Data at Rest

Most NoSQL databases are demanded upon the protection provided specially in data storage, just a slight NoSQL database categories that present protecting mechanisms for data at rest over using techniques of encryption. Encryption is widely regarded as the de-facto standard for safeguarding data in storage. Malicious intruders who intend to then steal from archives or with intention to read directly from the disk will find the data unintelligible. Encrypted data will be accessed by users with decryption keys, but however most industry solutions offering encryption services lack horizontal scaling and transparency required in the NoSQL environment (Kadebu & Mapanga, 2014).

2.3.7 Challenges in Enforcing Access Control

Clearly, it is difficult to enforce role-based access control in the NoSQL database's schema-less structure. For example, the Key-Value store that store data by means of a distributed index for object storage. In this type of database different data are stored in one huge database. This becomes a challenge as heterogeneous data is stored together in one database as opposed to relational models which conform to defined schemas and tables that store only related data.

2.3.8 Administrative Data Access

NoSQL database systems lack in-built facilities, documentation, and third-party tools to address issues of administrative rights with for instance full access to data and enabling creation of administrative boundaries for the purposes of encryption. Tasking administrators to choose the right security controls that would tighten all the screws on the four corners of our database systems like the proper access controls and the best encryption technologies can cause unwanted direct access to data files or data node processes. It will be better if this is left to the system designer to select controls to close this gap (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.9 Configuration and Patch Management

Existing configuration management tools work for underlying platforms. Different nodes or clusters of servers may have different patch revisions. Added nodes may have newer patches than existing nodes. This may create challenges in enforcing security uniformly across the NoSQL database environment.

2.3.10 Firewalls

Firewalls cannot protect data at rest or in-transit within the NoSQL database environment. If a firewall gets breached, the database is immediately exposed to attacks. Firewall breaches emanating from the firewall perimeter cannot be avoided like attackers who get into data centres physically or electronically can get access to data (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.11 Authentication Clients

Kerberos can be used to authenticate clients, DataNode, NameNode in the NoSQL database environment. Malicious Clients and Nodes can gain unauthorised access to the NoSQL database system upon stealing or duplicating the Kerberos ticket. These credentials can be obtained from system snapshots as well as virtual images. The situation has worsened in this Big Data environment where exact copies, clones and imposter nodes can be used to generate malicious services into the databases environment (Kadebu & Mapanga, 2014).

2.3.12 Audit and Logging

Audits and logs are performed to aid in discovery of malicious activities in the database system. However without actually looking at the data and developing policies to detect malicious activities, logging is not useful (Kadebu & Mapanga, 2014).

Also the frequency at which the Audits are carried out can have impact on their effectiveness. If audits are performed say quarterly that means malicious activities can occur which can result in serious problems for the organisation. This may be discovered too late when the damage has already occurred (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.13 Monitoring, Filtering, and Blocking

Existing NoSQL database monitoring tools lack the capability to identify malicious queries, misuse activities and subsequently block them. Monitoring undertaken by several tools in the NoSQL database environment mostly perform their task at the API.

There is an assumption that all access by client connections will pass through the same path that authenticate clients through Kerberos, which results in a performance constrains. Also advanced threats may bypass the central Kerberos authentication (Mapanga & Kadebu, 2013; Kadebu & Mapanga, 2014).

2.3.14 API security

APIs can be subjected to several attacks such as Code injection, buffer over flows, command injection as they access the NoSQL databases.

The APIs for big data clusters need to be protected from code and command injection, buffer overflow attacks, and every other web services attack. This responsibility often left to the application that uses the cluster (Kadebu & Mapanga, 2014).

2.4 NoSQL MongoDB Database

MongoDB is an open source NoSQL database schema-free product using a document-oriented data model and written in the C++ programming language. MongoDB was developed by MongoDB Inc. Software Company (formerly 10gen).

2.4.1 Overview

Mongod2 is the name of daemon process in NoSQL MongoDB. It deals with data requests, treats data formats, and manages background management operations. The main features are explained below:

1. Document-Oriented Storage

MongoDB supports flexible schemas and its collections do not enforce document structure like SQL databases – schemas do not have to be defined before inserting the data. This flexible schema nature makes accessing documents easy for adding an entity or an object.

Practically, the documents which exist in the same a collection has a common structure. MongoDB database saves its data in the format of documents, written in pairs of JSON-like field and value. Documents' structures in the form of key and value are similar to languages of software programming that links keys with values, whereas keys may have different of keys and values pairs (Murugesan, & Ray, 2014).

A JSON document might, for example, take all the data stored in a row that spans 20 tables in a relational database and aggregate it into a single document/object. Aggregating this information may lead to duplication of information, but this is not an issue as storage is no longer expensive.

The cost in storage is offset by the flexibility in the resulting model, ease of efficiently distributing the resulting documents, and read and write performance improvements, all of which are needed for web-based applications.

2. Full Index Support

In MongoDB where indexes are doing the same job as in other database systems such as improving the performance of repeatedly requested queries. Further, indexes are defined in MongoDB in the level of the collection and also MongoDB provides indexes in the level of the field as well the sub-field level in the documents under the collections. The objective of using the index in MongoDB is decreasing the documents number it must inspect in case of an appropriate index is used in a query, more on the above, MongoDB in some cases uses the data from the index to answer a query (Murugesan, Ray, 2014).

3. Replication and High Availability

A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. Replication process improves data availability and presents redundancy. By providing several database servers that have multiple copies of data, replication contributes in protecting a database in case of losing a single server.

In addition, replication provides the advantage of recovery in service interruptions or hardware failure. As well, other advantages such as additional copies of the data, disaster recovery, reporting, and backup are all facilitated (Murugesan & Ray, 2014).

4. Auto-Sharding

Sharding is useful in the situation where is the volume of data is huge and an individual machine cloud not be enough to save the data as well it could not be capable of presenting a sufficient read and write throughput. Accordingly, sharding provides the answer for this question.

Sharding process is implemented by storing data across more than one machines. With sharding, problem can be solved horizontally by adding additional machines to face the challenge of growing data and the demands of read and write processes.

MongoDB achieves scaling by auto-sharding. MongoDB sharding provides: (1) automatic balancing for changes in load and data distribution, (2) easy addition of new machines without down time, (3) no single points of failure, and (4) automatic failover (Liu, Wang & Jin, 2012; Murugesan & Ray, 2014).

5. Querying Mode Data

Querying Mode Data is manipulated through CRUD (create, read, update, and delete) operations. MongoDB provides rich semantic querying options for reading data. For example, the method of MongoDB “db.collection.find()” returns documents of a certain a collection. Moreover, The method db.collection.find() retrieves a cursor to the returned documents (Liu, Wang & Jin, 2012; Murugesan & Ray, 2014).

6. Fast In-Place Updates

MongoDB allows speedy update operations through atomic modifiers. Database updates are fast in spite of the data being spread across hundreds of servers. In MongoDB, methods; db.collection.update() and db.collection.save() update actual documents existing in a collection. The db.collection.update() presents more control over the updating process (Murugesan & Ray, 2014).

Below are the essential features of MongoDB:

1. MongoDB data model*:*

regarding to the data model, MongoDB consists of collections. A collection which is schema-free. This collection matches a table in relational databases as it shows in Table 2.3. Moreover, a collection has documents which is as a row in a collection and each document has an id (MongoDB, 2014).

Table 2.3

MongoDB vs SQL terms

MongoDB Database Term	**SQL database Term**
Database	Database
Collection	Table
Document	Row
Field	Column
Index	Index
Id	Primary Key
Embedding and linking	Join

2. MongoDB API:

Mongo Query Language is a query language for MongoDB. To deal with documents and collection, a query is built including the certain document or collection that the needed by the query. Also, RESTful as an API in MonogDB, REST stands for (Representational State Transfer) is designed architecture for building networked applications.

3. MongoDB Architecture:

MongoDB as cluster is based on sharding technique. A shard automatically saves a part of the data. Input/Output processes are automatically leaded to the appropriate shard(s). Every shard has a replica set which can a server or more having copies of the same data. At any certain time

4. MongoDB replication:

Replica-set and Master-Slave are two types of replication for MongoDB. Both types common in writing operation which is implemented on one server (Master or Primary).

5. Sharding:

MongoDB provide the functionality of sharding based on automated architecture for sharding/partitioning (MongoDB, 2014).

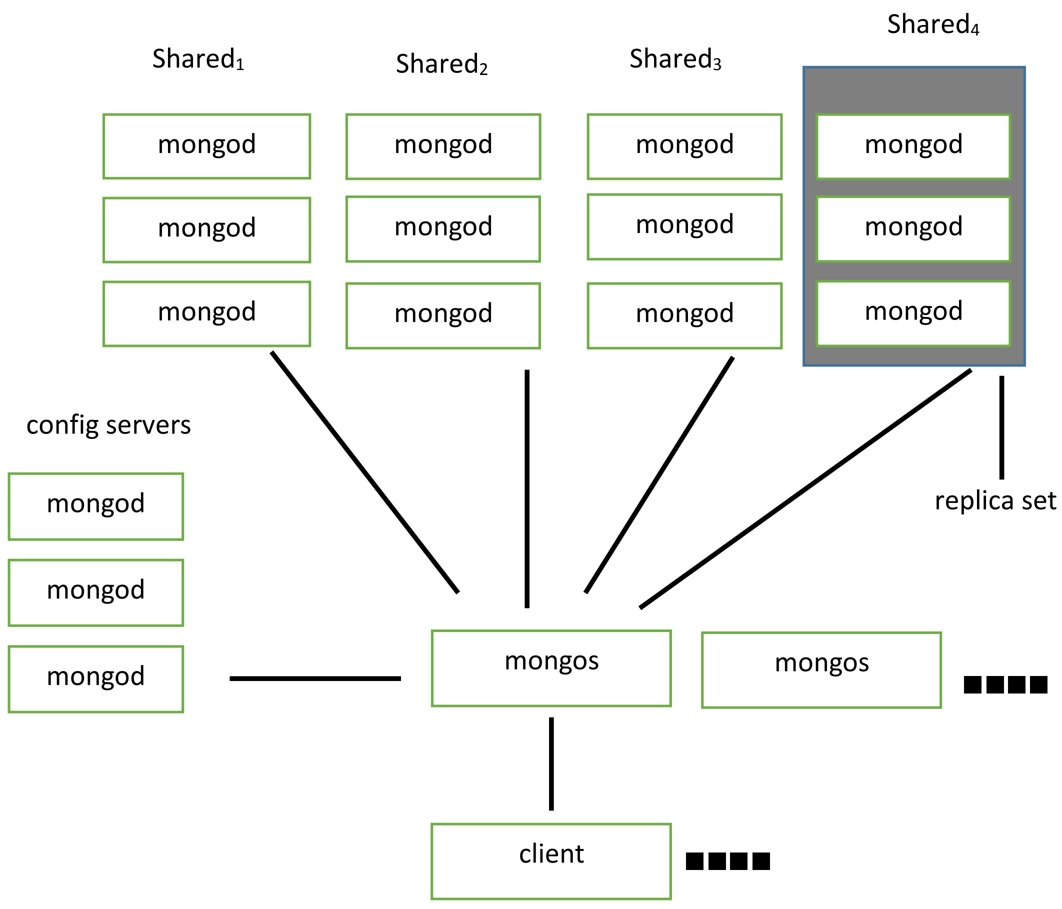


Figure 2.1 MongoDB Architecture (Okman, Gal-Oz, Gonen, Gudes, & Abramov, 2011)

One is primary server and the remaining are secondary. If the primary goes down one of the secondary servers works automatically as primary.

2.5 NoSQL Database Auditing

2.5.1 Database Auditing Definition

Database auditing is the process of monitoring access to and modification of selected database objects and resources within operational databases and retaining a detailed record of the access where said record can be used to proactively trigger actions and can be retrieved and analysed as needed (Mullins, 2014).

2.5.2 Importance of NoSQL Database Auditing

NoSQL auditing is a crucial aspect in NoSQL database security. Actually, organizations should provide measures of database monitoring or auditing in order to avoid putting their valuable assets exist in database under high risk. Also, organizations are under pressure to protect sensitive information and implementing database auditing is considered the only solution to this problem (Liu & Huang, 2009).

Table 2.4

Auditing Types and Descriptions

Type of Auditing	**Meaning/Description**
Statement Auditing	Enables user to audit SQL statements by type of statement, not by the specific schema objects on which they operate. Typically broad, statement auditing audits the use of several types of related actions for each option. For example, AUDIT TABLE tracks several DDL statements regardless of the table on which they are issued. You can also set statement auditing to audit selected users or every user in the database.
Privilege Auditing	Enables user to audit the use of powerful system privileges that enable corresponding actions, such as AUDITCREATE TABLE. Privilege auditing is more focused than statement auditing, which audits only a particular type of action. You can set privilege auditing to audit a selected user or every user in the database.

Schema Object Auditing	Enables user to audit specific statements on a particular schema object, such as AUDIT SELECT ON employees. Schema object auditing is much focused, auditing only a single specified type of statement (such as SELECT) on a specified schema object. Schema object auditing always applies to all users of the database.
Fine-Grained Auditing	Enables user to audit at the most granular level, data access and actions based on content, using any Boolean measure, such as value > 1,000,000. Enables auditing based on access to or changes in a column.

The important phase of NoSQL database auditing is logging. The logging process aims to record database activities and other database information such as system performance and use on administrator's demand. Then, the logs can be utilized in audit trail analysis and database usage report generation, so as to (Liu & Huang, 2009):

1. Discover any violations of database security policies.
2. Determine if there are attacks to the database.
3. Asset in database recovery.

2.5.3 NoSQL DBMS Auditing

Auditing is an instrument of the NoSQL DBMS that facilitate the process of tracking the usage of database resources and authority enables by DBAs (Geer, 2005). In situations where auditing is available and provided the DBMS will generate an audit trail recording target database operations. Each audited database operation has records in an audit trail with information such as; name of database object, operation performed when and by who, based on the level of auditing available in the DBMS, logs of changes happened on an actual record of data also may be recorded. But it has some limited functionality to predict the attacks.

It is very useful to find that what type of operation has been made (Ezumalai & Aghila, 2009).The most common technique for audit storage is to record all the DBMS audit data in a single audit trail. This audit trail can be inside an audit database or a stand-alone file managed by the OS. This technique helps also in the audit analysis task since all the audit data is sequentially saved in a one place, which enable any analysis tool to search easily correlate related events. The selection of audit analysis tools (range between primitive and sophisticated) determines the ease of analysis which can range from the (Wisseman, Wilson, & Wichers, 1996).

Table 2.5

Auditing in NoSQL Databases

NoSQL Database	**Auditing Status**
MongoDB	Not available except for DDL and replica data
Cassandra	Enterprise Edition only
HBase	Not available
CouchDB	Not available
Couchbase Server	Not available
Neo4J	Not available
Amazon SimpleDB	Not available

Table 2.5 is exploring the auditing as a security aspect in NoSQL databases. The data explained that some of them do not have auditing feature at all such as CouchDB, Couchbase Server, Neo4J and Amazon SimpleDB. Moreover, the table also shows that some NoSQL databases provide auditing but with limited capabilities (Grolinger, Higashino,Tiwari & Capretz, 2013).

2.5.4 MongoDB Database Auditing

Currently, MongoDB Enterprise new version 2.6 includes an auditing capability. The auditing facility allows administrators and users to track system activity for deployments with multiple users and applications. The auditing facility can write audit events to the console, the syslog, a JSON file, or a BSON file. The current auditing system in MongoDB can only record DDL in addition to replica set, authentication and authorization, and general operations.

MongoDB auditing system records the following actions that are related to DDL operations:

createCollection, createDatabase, createIndex, renameCollection, dropCollection, dropDatabase, dropIndex, createUser, dropUser, dropAllUsersFromDatabase, updateUser, grantRolesToUser, revokeRolesFromUser, createRole, updateRole, dropRole, dropAllRolesFromDatabase, grantRolesToRole, revokeRolesFromRole, shutdown, grantPrivilegesToRole, revokePrivilegesFromRole and shardCollection (MongoDB, 2014).

Table 2.6

Examples of DML/CRUD operations (Truică, Boicea, & Trifan, 2013)

Operation	**SQL-Like (MYSQL)**	**MongoDB**
Insert	INSERT INTO USERS(id, first_name, last_name) VALUES (1, "Ciprian", "Truica")	db.articles.insert({ _id: "1", age: 45, status: "A" })
Select	SELECT * FROM USERS	db.articles.find()
Select fields	SELECT frist_name, last_name STATUS FROM USERS	db.articles.find({ }, { first_name: 1, last_name: 1 })
Select with where	SELECT u.first_nameFROM `BlogDB`.`users` AS uWHERE u.id = 1;	db.articles.find({user_id:"1
Ordered Select ASC	SELECT * FROM USERS ORDER BY USER_ID ASC	db.articles.find({}).sort({user_id : 1})
Ordered Select DESC	SELECT * FROM USERS ORDER BY USER_IDDESC	db.articles.find({}).sort({user_id: -1 })
Select with count	SELECT COUNT(*) FROM USERS	db.articles.count()

Update	UPDATE \`BlogDB\`.\`articles\` SET title="MongoDB" WHERE id = 1;	db.articles.update({_id: "1"}, $set : { "article.title": "MongoDB" }}, {upsert: true});
Delete	DELETE FROM USERS	db.articles.remove( )
Delete using where	DELETE FROM USERS WHERE id=”1”	db.articles.remove({ _id: "1" })

2.6 Related Work

As NoSQL trend is relatively new, there are various opportunities for research and development. Many researchers are attracted to this category of databases. Apart from other NoSQL databases we discuss here the work done in MongoDB as follows. The security features and main functionality of the most two popular NoSQL databases: Cassandra and MongoDB are reviewed by Okman, Gonen and Abramoy (2011). The study found that the common problem to both NoSQL databases is lack of data files encryption, simple authentication among servers and between them and the client, authorization doesn’t support fine-grained authorization or Role Based Access Control (RBAC), and lack of auditing. This gives the opportunity to implement the security aspects in MongoDB like NoSQL databases.

The basic map-reduce algorithm is studied and it is claimed that the NoSQL databases such as MongoDB and its key-value stores provide an efficient framework to aggregate large volumes of data (Bonnet, Laurent, Sala, Laurent, & Sicard, 2011). The comparison between Oracle and MongoDB is done by considering various issues such as theoretical, differences, features, restrictions, integrity, distribution, system requirements and architecture, query and insertion times.

They stated that MongoDB provides flexibility, horizontal scalability. Also it can store complex data like array, object or reference into one field. Mapping of objects is also very easy in MongoDB. The features of MongoDB like map-reduce and replications of data make the development faster than the Oracle. Being open source, plug-ins for MongoDB can be developed for easy work (Boicea, Radulescu, & Agapin, 2012).

Van der Veen, Van der Waaij, and Meijer (2012) studied the performance difference among SQL, NoSQL databases such as one open source SQL database (PostgreSQL) and two open source NoSQL databases (Cassandra and MongoDB) with regard to the sensor data storage. It is shown that MongoDB is the best choice for a small or medium sized non-critical sensor application, especially when write performance is important.

In the research conducted by Liang and Mizuno (2011), it is stated that it is necessary to detect defects in the code at early stage to assure the quality of the software which can be achieved by using code review activity. The researchers have analysed the code review repository of open source software, Chromium with MongoDB as the back end. Before that they addressed seven research questions for which they found interesting answers.

The principles and implementation mechanism of Auto-Sharding in MongoDB along with the improved algorithm based on the frequency of data operation is considered by Liu, Wang and Jin (2012). They claimed that this algorithm improves the concurrent read and write performance of cluster by effectively balancing the data among shards.

The study was performed by Rutishauser (2012) in which TPC-H queries were implemented in MongoDB to see the performance difference with the open source RDBMS PostgreSQL. In his paper he found that the performance of the MongoDB was very poor as compared to the PostgreSQL. The effort is carried out to improve the performance of web services interactions (Zagarese, Canfora, Zimeo & Baude, 2012). They determined the execution contexts quantitatively, that make dynamic offloading effective.

In summary, sample of MongoDB related work has been shown in Table 2.7. The table presents a brief information about authors, year of publication, title and findings of the researches column in the table depicts the names of the researchers. This related work sorted by the year of publication.

Table 2.7

Sample of MongoDB Related Work

Author	**Year**	**Title**	**Brief/Findings**
Okman, Gonen and Abramoy	2011	Security Issues in NoSQL Databases	The main functionality and security features of two of the most popular NoSQL databases: Cassandra and MongoDB
Bonnet, Laurent, Sala, Laurent and Sicard	2011	Reduce, you say: What NoSQL can do for data aggregation and bi in large repositories	The basic map-reduce algorithm and it is claimed that the NoSQL databases such as MongoDB and its key-value stores provide an efficient framework
Liang and Mizuno	2011	Analyzing Involvements of Reviewers Through Mining A Code Review Repositor	Chromium with MongoDB as the back end. Before that they addressed seven research questions for which they found interesting answers.
Boicea, Radulescu and Agapin	2012	MongoDB vs Oracle-Database Comparison	The features of MongoDB like map-reduce and replications of data make

			the development faster than the Oracle
Van der Veen, Van der Waaij, and Meijer	2012	Sensor data storage performance: Sql or nosql, physical or virtual. In Cloud Computing (CLOUD)	It is shown that MongoDB is the best choice for a small or medium sized non-critical sensor application, especially when write performance is important.
Kadebu and Mapanga	2014	A Security Requirements Perspective towards a Secured NOSQL Database Environment	Identified NoSQL major security issues including audit and logging, monitoring, filtering, blocking and API security.

2.7 Conclusion

The chapter reviewed NoSQL database and its data models. In addition, NoSQL database auditing, its definition and importance have been discussed. Moreover, NoSQL DBMS auditing was explored. Lastly, it described in details MongoDB auditing and MongoDB related works were presented.

Accordingly and upon this literature review, it is clear that MongoDB database company tries to mitigate the auditing gap by providing MongoDB new enterprise version 2.6 (8th of April 2014). This version included an auditing system which can record DDL operations as it was cleared from the above.

But unfortunately, it still unable to record Data Manipulation Language (DML) and most organizations require that all user-based Data Manipulation Language (DML) and/or CRUD (Create, Read, Update and Delete) operations performed against production databases be logged. This request extends the MongoDB auditing framework, introduced in version 2.6, to include logging of all user queries and DML/CRUD operations.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

This chapter describes the research methodology used for this study. It defines the research methods and how they will help to answer the posed research questions.

3.2 Research Methodology

This research aims to improve the auditing functionality in MongoDB. In order to achieve that a new auditing mechanism was proposed using a research methodology with three stages and each stage has sub-category phases as it is explained in Figure 3.1.

Figure 3.1 shows the study methodology which consists of three stages. Firstly, stage one has one phase focus on usage of literature review in order to achieve the first objective. Secondly, stage two contains three phases; building the mechanism architecture, creating the extracting algorithm and developing a prototype. Eventually, stage three consists of also three phases; conducting experiment, obtaining results and discussing these results. The stages of the methodology elaborated in details in the following sections.

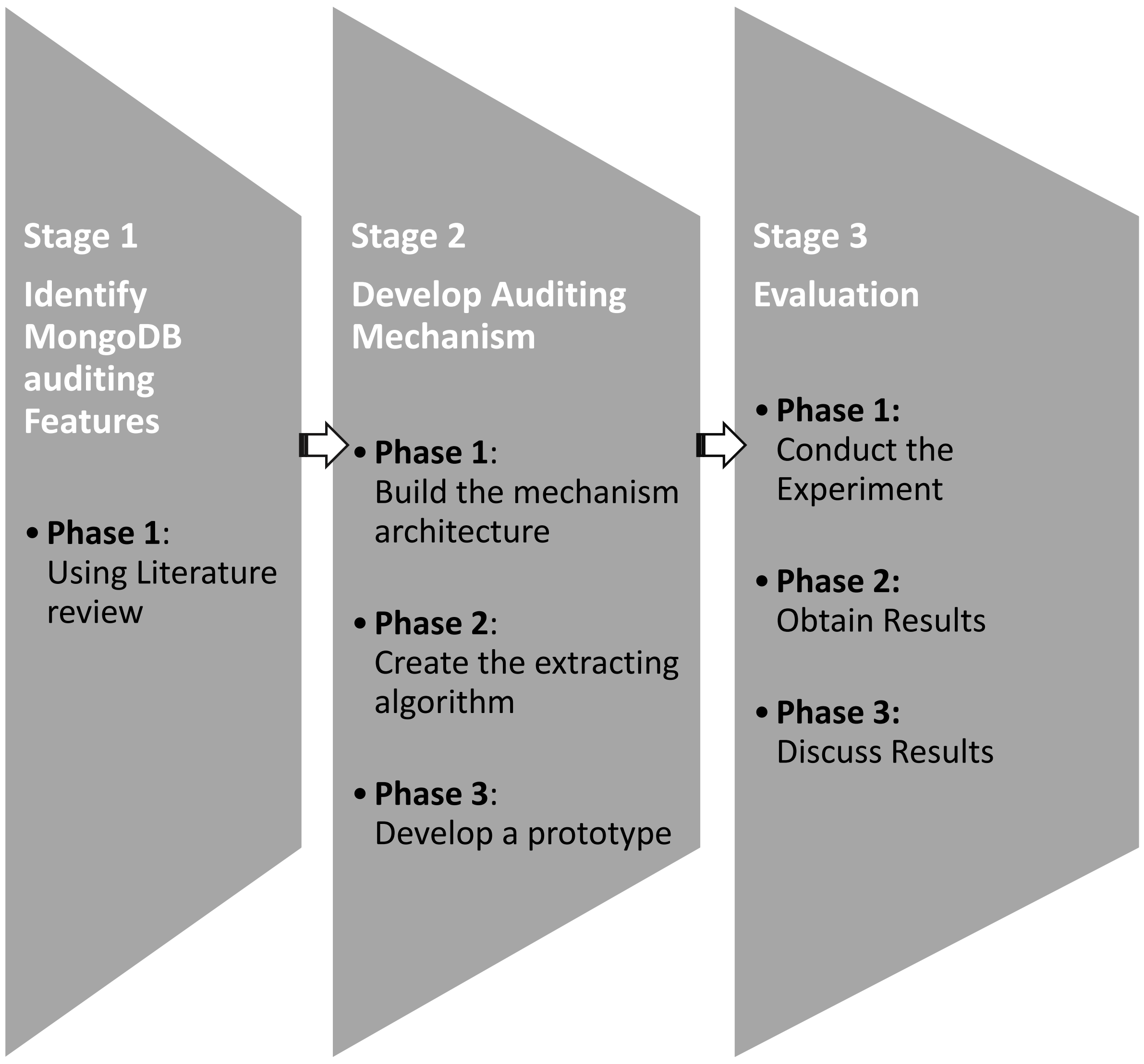

Stage 1
Identify MongoDB auditing Features
• Phase 1: Using Literature review
Stage 2
Develop Auditing Mechanism
• Phase 1: Build the mechanism architecture
• Phase 2: Create the extracting algorithm
• Phase 3: Develop a prototype
Stage 3
Evaluation
• Phase 1: Conduct the Experiment
• Phase 2: Obtain Results
• Phase 3: Discuss Results

Figure 3.1 Stages of research methodology

3.2.1 Stage 1: Identifying MongoDB auditing features

Phase 1:

This phase depends on literature review to identify MongoDB auditing features. The main focus is determining current auditing functionalities. Initially, this phase starts with collecting literatures, analysing it and eventually determining the existing MongoDB auditing features. The main resources for this phase including researches in addition to MongoDB official website.

3.2.1.1 Analysis of Logging Techniques in MongoDB

Mongodb gives various options for storing logs that are discussed below.

1. Oplog (operations log) is used mostly to propagate updates in a distributed setting. Oplog is a dedicated capped collection responsible for keeping a log of all operations that change the existed data. It represents the base of replication process in MongoDB.

Oplog contains a log of all the write operations occurring in MongoDB. Database operations are applied on the primary first in MongoDB and sequentially logs them on the oplog. Then secondaries listen to primarys oplog and then duplicate and implement these operations in an asynchronous manner. Moreover, a copy of the oplog is available in all replica set members, enabling them to manage the existing case of the database.

Each replica set member can extract oplog entries from any other member. Each operation in the oplog is idempotent, so applying an operation multiple times creates no problems. Oplog can be queried like any other collection, but works only in the presence of replication (Murugesan, Ray, 2014).

2. Diaglog provides a verbose diagnostic log that records database transactions and operations. It does not record read and insert operations, although update and authentication information are recorded. Diaglog generates a more detailed log that is very helpful in the problem solving and logs records for certain errors. MongoDB's dbpath directory is place where it keeps these log files in. the naming convention for these files is diaglog dot time in hexadecimal (Murugesan, Ray, 2014).

3. Verbose logging allows logs to be stored in user specified path by choosing –logpath option when running it in the background. Verbose logging will make logs quite large and may affect server performance. To append to an existing log rather than overwriting it, mongod process can be started with the –logappend option. Finally, if a MongoDB process is long-running, logrotate command can be used to move logs to new file. Verbose logging helps in recording every update operation, but not insert and read operations. It records authentication information as well.

4. Profiler Audit logs can be extracted using Profiler which records complete set of all database queries executed by different users. They record read, update and other user control operations but their insert recording is unclear and also access control information are not properly stored (Murugesan, Ray, 2014).

5. Mongosniff provides a low-level operation tracing/sniffing view into database activity in real time. This is a MongoDB specific analogue of tcpdump for TCP/IP network traffic. It helps in recording only write operation including insert and update operations. It is very evident that each technique individually does not record all the operations. Profiler does most of the required job for log management by recording read, update, and authentication information (Murugesan, Ray, 2014).

3.2.1.2 MongoDB Auditing Features:

The auditing facility in MongoDB Enterprise new version 2.6 allows administrators and users to track system activity for deployments with multiple users and applications. The auditing facility can write audit events to the console, the syslog, a JSON file, or a BSON file. The current auditing system in MongoDB can only log DDL operations in addition to information related to replica set. Likewise, the current auditing system also can record authentication and authorization processes, and general operations (MongoDB, 2014).

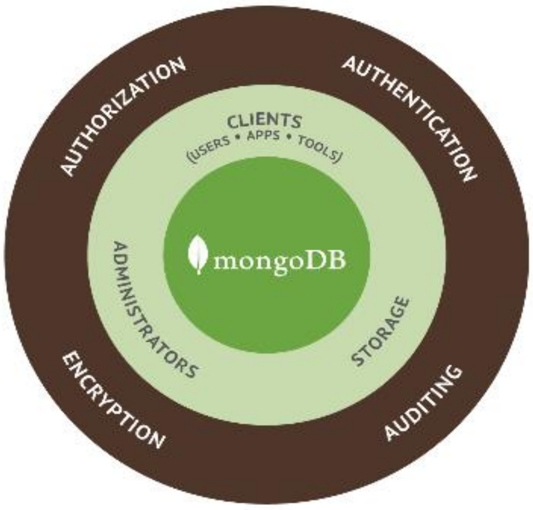


Figure 3.2 MongoDB Security Architecture (MongoDB, 2014)

3.2.1.3 Auditing Events and Filters

The auditing system can record the following operations:

• Data Definition Language (DDL).

• Replication.

• Authentication and authorization processes, and general operations.

Table 3.1

Description of event message fields

Field	**Type**	**Description**
atype	String	Action type
ts	Document	Document that contains the date and UTC time of the event, in ISO 8601 format.
local	Document	Document that contains the local IP address and the port number of the running instance.
remote	document	Document that contains the remote IP address and the port number of the incoming connection associated with the event.
users	array	Array of user identification documents. Because MongoDB allows a session to log in with different user per database, this array can have more than one user. Each document contains a user field for the username and a database field for the authentication database for that user.
params	document	Specific details for the event
result	integer	Error code.

Table 3.1 describe description of event message fields with details (MongoDB, 2014). As it is cleared the message includes fields with certain data type such as atype which represents the action type with string as a data type and users' field with array as a data type that refer to the array of user identification documents.

Table 3.2

MongoDB auditing system records the following Operations

Operations Type	
Authenticate	createUser
authCheck	dropUser
createCollection	dropAllUsersFromDatabase
createDatabase	updateUser
createIndex	grantRolesToUser
renameCollection	revokeRolesFromUser
dropCollection	createRole
dropDatabase	updateRole
dropIndex	dropRole
revokeRolesFromRole	dropAllRolesFromDatabase
grantPrivilegesToRole	grantRolesToRole
shardCollection	revokePrivilegesFromRole
addShard	replSetReconfig
removeShard	enableSharding
shutdown	applicationMessage

MongoDB auditing system records the operations that are related to DDL operations. Table 3.2 contains the DDL operation types that can be recorded by MongoDB. These types focus on the actions happen to users, database, collection and role in addition to replica set, authentication and authorization. For example, "createUser" operation records information about creating new user. As well, "createCollection", "renameCollection" and "dropCollection" record information about creating new collection, renaming a collection and dropping a collection respectively.

3.2.2 Stage 2: Develop Auditing Mechanism

The target of stage two to establish an auditing mechanism for MongoDB. This stage consists of three sub-phases; Designing the auditing architecture, creating the algorithm and prototype for the architecture and evaluating the mechanism respectively.

Phase 1: Build the architecture for the mechanism to extract DML/CRUD operations from MongoDB.

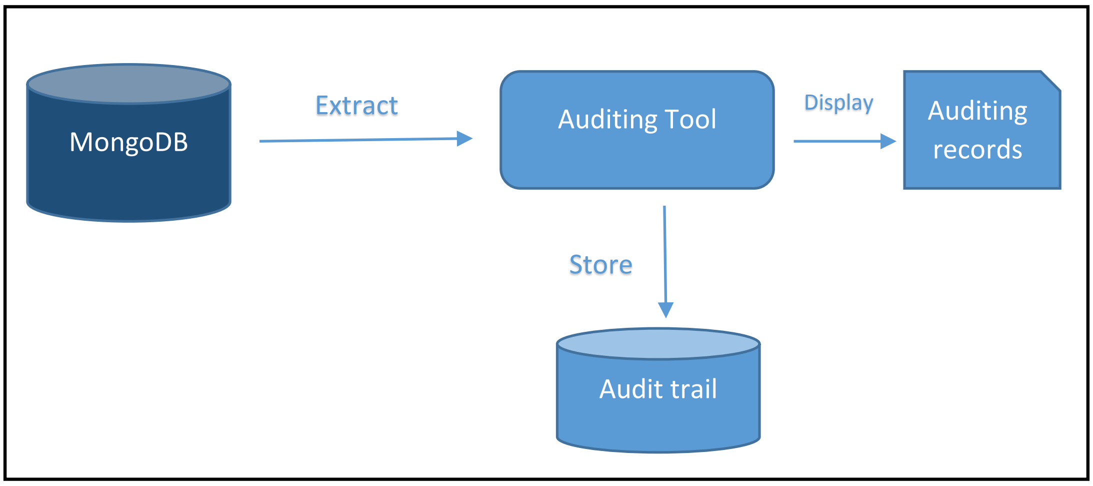
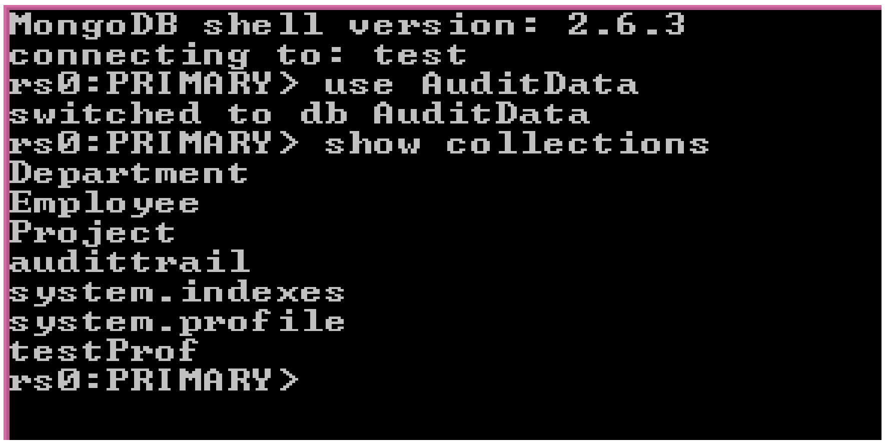


Figure 3.3 The architecture of auditing mechanism

Figure 3.3 above describes the components of the proposed auditing mechanism. The mechanism consists of four components MongoDB, auditing tool, auditing records and auditing trail which will be explained in details in next section.

1. MongoDB:

The essential part and represents the auditing database in the proposed mechanism.



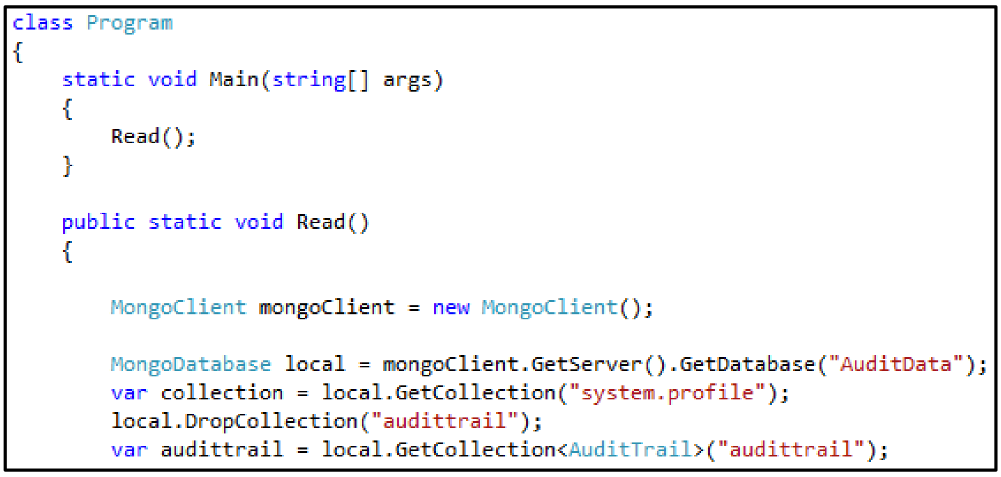
```
MongoDB shell version: 2.6.3
connecting to: test
rs0:PRIMARY> use AuditData
switched to db AuditData
rs0:PRIMARY> show collections
Department
Employee
Project
audittrail
system.indexes
system.profile
testProf
rs0:PRIMARY>
```

Figure 3.4 MongoDB Database

Figure 3.4 shows MongoDB after it is installed and run. As it cleared from the figure MongoDB has a shell version. So, with dedicated shell script MongoDB can be opened and accessed easily.

2. Auditing Tool:

Auditing tool denotes a software program written in C# using Visual Studio under the .Net 4.0 framework and responsible for extracting DML/CRUD operations from MongoDB, storing them in an audit trail and display them. The Auditing tool programming code depends on the auditing algorithm. Furthermore, the main function for this tool is extracting the DML/CRUD operations from MongoDB by accessing the logging files exist in MongoDB. Logging techniques and files has been determined in stage one upon the research methodology as mentioned above.



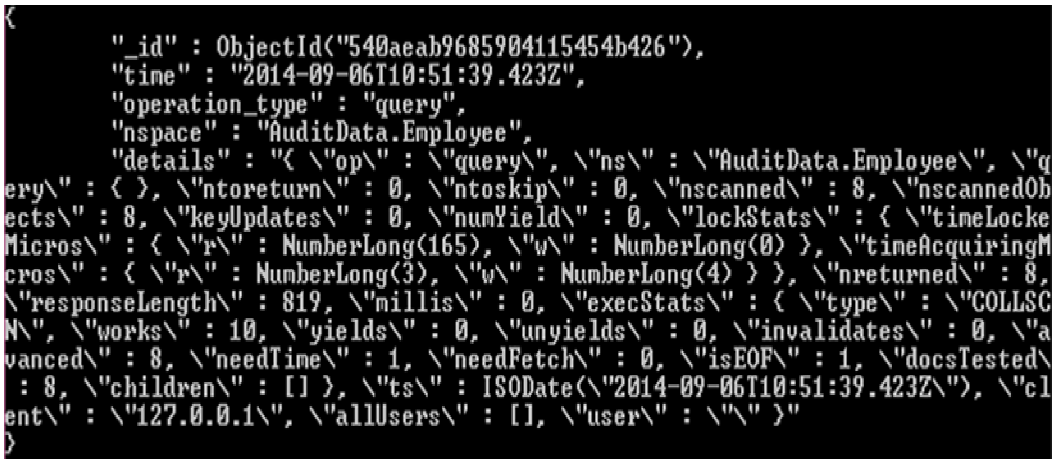
```
class Program
{
    static void Main(string[] args)
    {
        Read();
    }

    public static void Read()
    {

        MongoClient mongoClient = new MongoClient();

        MongoDatabase local = mongoClient.GetServer().GetDatabase("AuditData");
        var collection = local.GetCollection("system.profile");
        local.DropCollection("audittrail");
        var audittrail = local.GetCollection<AuditTrail>("audittrail");
```

Figure 3.5 Sample of C# code

Figure 3.5 above shows part of the implementation C# code of the proposed algorithm. The first line is the name of the class which is called “Program” and includes two public methods; first one is called “main” and the second is called “read”. The first method “main” call the second method “read” and the second method includes code for creating MongoDB client and connect to the database then retrieve the target data.

3. Audit Trail:

Audit trail denotes a log contains DML/CRUD records that are extracted from MongoDB by the audit tool. This trail actually exists in the same audited MongoDB database. The audit trail contains recorded data about DML/CRUD operations needed to be auditing.



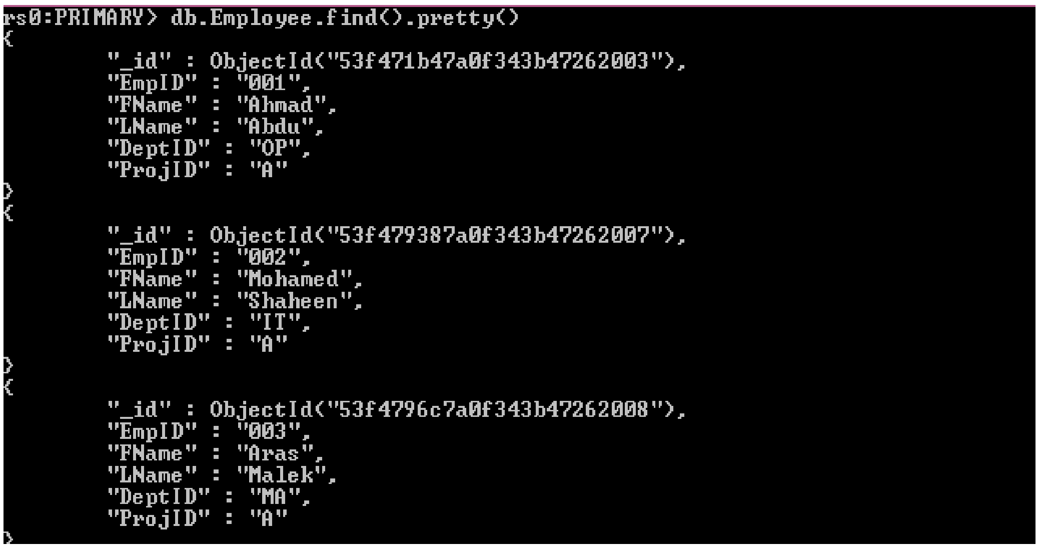
```
{
        "_id" : ObjectId("540aeab9685904115454b426"),
        "time" : "2014-09-06T10:51:39.423Z",
        "operation_type" : "query",
        "nspace" : "AuditData.Employee",
        "details" : "{ \"op\" : \"query\", \"ns\" : \"AuditData.Employee\", \"q
ery\" : { }, \"ntoreturn\" : 0, \"ntoskip\" : 0, \"nscanned\" : 8, \"nscannedOb
ects\" : 8, \"keyUpdates\" : 0, \"numYield\" : 0, \"lockStats\" : { \"timeLocke
Micros\" : { \"r\" : NumberLong(165), \"w\" : NumberLong(0) }, \"timeAcquiringM
cros\" : { \"r\" : NumberLong(3), \"w\" : NumberLong(4) } }, \"nreturned\" : 8,
\"responseLength\" : 819, \"millis\" : 0, \"execStats\" : { \"type\" : \"COLLSC
N\", \"works\" : 10, \"yields\" : 0, \"unyields\" : 0, \"invalidates\" : 0, \"a
vanced\" : 8, \"needTime\" : 1, \"needFetch\" : 0, \"isEOF\" : 1, \"docsTested\
 : 8, \"children\" : [] }, \"ts\" : ISODate(\"2014-09-06T10:51:39.423Z\"), \"cl
ent\" : \"127.0.0.1\", \"allUsers\" : [], \"user\" : \"\" }"
}
```

Figure 3.6 Sample of audit trail records

The data appeared in Figure 3.6 shows the record id, time, operation type, name space and details respectively. These details are available for each DML/CRUD recorded operation.

4. Displaying Auditing Records:

Auditing records stored in audit trail is displayed by the auditing tool. The Auditing tool access the audit trail existed in MongoDB and display the recorded DML/CRUD operations.



```
rs0:PRIMARY> db.Employee.find().pretty()
{
        "_id" : ObjectId("53f471b47a0f343b47262003"),
        "EmpID" : "001",
        "FName" : "Ahmad",
        "LName" : "Abdu",
        "DeptID" : "OP",
        "ProjID" : "A"
}
{
        "_id" : ObjectId("53f479387a0f343b47262007"),
        "EmpID" : "002",
        "FName" : "Mohamed",
        "LName" : "Shaheen",
        "DeptID" : "IT",
        "ProjID" : "A"
}
{
        "_id" : ObjectId("53f4796c7a0f343b47262008"),
        "EmpID" : "003",
        "FName" : "Aras",
        "LName" : "Malek",
        "DeptID" : "MA",
        "ProjID" : "A"
}
```

Figure 3.7 Sample of data recorded in MongoDB database.

Figure 3.7 represents a sample of data recorded in MongoDB. Data structure in MongoDB is different from traditional RDBMS such as MySQL. In the figure there are three documents (a document matches a record in relational databases), each of them has six fields (field matches column in Relational database).

Phase 2: Create the algorithm for extracting DML/CRUD operations

The phase of creating an algorithm for extracting DML/CRUD operations from MongoDB is crucial due to establishing the auditing mechanism. The main function for this algorithm is getting the target DML/CRUD operations.

Accordingly, getting the required operations come through certain steps comprise accessing the MongoDB logs, extracting the DML/CRUD operations for assigned data and then storing them in the audit trail which is existed in the same MongoDB.

The Auditing Algorithm:

```
1-Connecting to MongoDB
2-Accessing the MongoDB log files
  2.1-While there are DML operations for the auditing data do the following:
        2.1.1- Extracting DML operations,
        2.1.2 - Transform their format to be more readable
        2.1.3 - Save them in the Audit Trail collection
        2.1.4 - Repeating the above steps until exit.
```

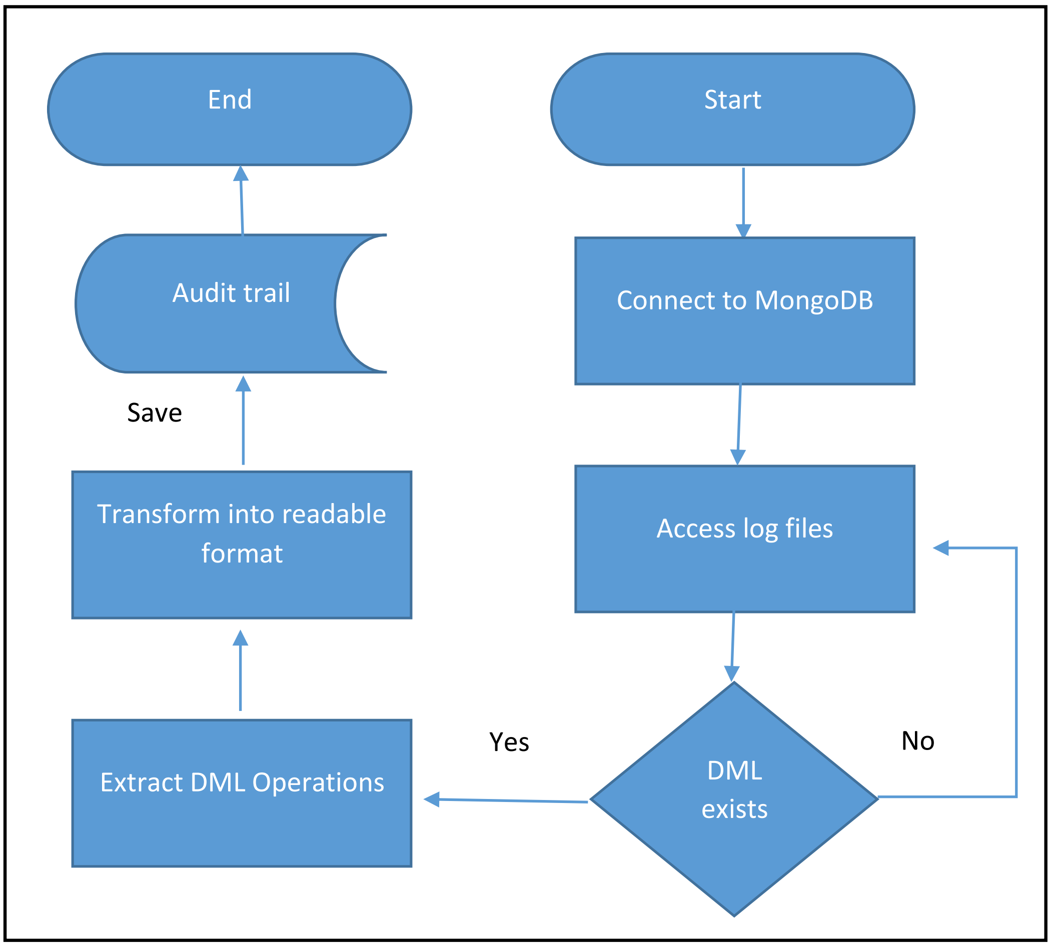


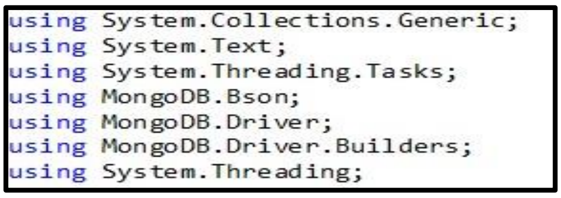
Figure 3.8 MongoDB auditing algorithm flowchart

The flowchart in Figure 3.8 above shows the sequence of the algorithm starting from connecting to MongoDB, accessing log files, checking for existence of DML/CURD operations, extracting the targeted operations, transforming to readable format and eventually save in the audit trail.

Phase 3: Develop a prototype for the mechanism

The phase three of stage two denotes building the prototype for the auditing algorithm. The prototype written in C sharp under Microsoft Visual Studio 2013 Desktop on a machine running an Intel i5 quad core 2.6GHz processor with 8GB of DDR3 Ram with Windows8 64bit operating system. In order to connect to MongoDB, the suitable

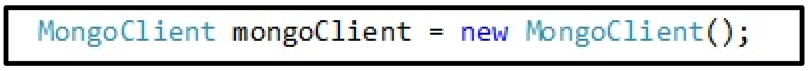
C sharp driver has been used. Developing process includes writing programming code to implement the auditing algorithm and achieve the research objective. Furthermore, appropriate libraries has been used as it is showed in Figure 3.9 to connect with MongoDB and accessing log files.



```
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Builders;
using System.Threading;
```

Figure 3.9 Libraries used in the C# code

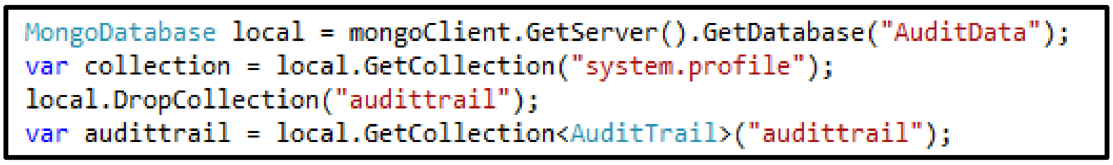
Extracting DML/CRUD operations, transforming them into readable format and lastly storing them in the audit trail. As it is cleared from the prototype programming code in C# the code starts by calling the appropriate libraries due to access MongoDB database correctly. Figure 3.10 shows that after the libraries the code begins to create a MongoDB client to connect.



```
MongoClient mongoClient = new MongoClient();
```

Figure 3.10 MongoDB client to access data

Sequentially, the code accesses the target data and then the profile log file and audit trail. After extracting DML records and transforming the data is saved in the audit trail as it is showed in Figure 3.11.



```
MongoDatabase local = mongoClient.GetServer().GetDatabase("AuditData");
var collection = local.GetCollection("system.profile");
local.DropCollection("audittrail");
var audittrail = local.GetCollection<AuditTrail>("audittrail");
```

Figure 3.11 Extracting and transforming C# code

3.2.3 Stage 3: Evaluation

To evaluate the proposed mechanism an experiment was conducted on two data sets to display the auditing of CRUD operations with the new auditing mechanism in MongoDB and measure the time taken when using it.

The experiment objectives:

- Examine the types of operations that can be audited (Insert – Select – Update – Delete).
- Evaluate the performance before and after applying the proposed auditing mechanism by calculating the time taken for executing CRUD/DML operations (Li & Manoharan, 2013).

1. **Phase 1: Conduct the Experiment:**

Table 3.3 shows the DML/CRUD operations that have been involved in the experiment involve. These operations are create/insert, query/select, update and remove/delete.

Table 3.3

DML/CRUD operations in MongoDB

Operation	**MongoDB**
Insert	db.articles.insert({ _id: "1", age: 45, status: "A" })
Select	db.articles.find({ }, { first_name: 1, last_name: 1 })
Update	db.articles.update({_id: "1"}, $set : { "article.title": "MongoDB" }}, {upsert: true});
Delete	db.articles.remove({ _id: "1" })

Experiment Setup:

Hardware and software: For the initial setup MongoDB was installed on a machine running an Intel i5 quad core 2.6GHz processor with 8GB of DDR3 RAM with Windows8 64bit operating system. On the MongoDB we use the latest version 2.6.

Data Sets:

Two data sets were used to conduct the experiment "Auditdata" and "Auditdata2". Auditdata includes three collection Department, Employee and Project. Likewise, AuditData2 includes three collections Lecturer, Course and Student. Table 3.4 shows these data sets in details.

Table 3.4

Description of data sets used in the evaluation stage

Data Set	Collection	# of Documents	Data Type
AuditData	Department	100000	Text and numbers
	Employee	100000	Text and numbers
	Project	100000	Text and numbers
AuditData2	Lecturer	100000	Text and numbers
	Course	100000	Text and numbers
	Student	100000	Text and numbers

Phase 2: Obtain the Results

The phase aims to getting and recording results of the experiment in terms of analysing them easily. More details and explanation can be seen in Chapter Four.

Phase 3: Discuss the Results

In this phase, results of the experiment will be discussed. More details and explanation can be seen in Chapter Four.

3.3 Conclusion

This chapter presents the research methodology stages which involved three stages and each stage has sub-phases. The main stages are; identification of MongoDB auditing features, developing auditing mechanism and evaluation of auditing mechaism and all of these stages have been achieved. Moreover, the outcome of the first stage was related to MongoDB auditing features and shows that MongoDB auditing system logs operations information including; schema data definition language operations and operations related to replica set in addition to operations of authentication and authorization, and eventually general operations.

After that, in the phase one of stage two a new mechanism architecture was presented. The auditing mechanism has four components one of them was the auditing tool. Algorithm was developed for this auditing tool in phase two of stage two. In order to validate this algorithm a prototype was built. Morover, last stage used in evaluation process and more discussion is made in upcoming chapter.

CHAPTER FOUR

RESULTS AND EVALUATION

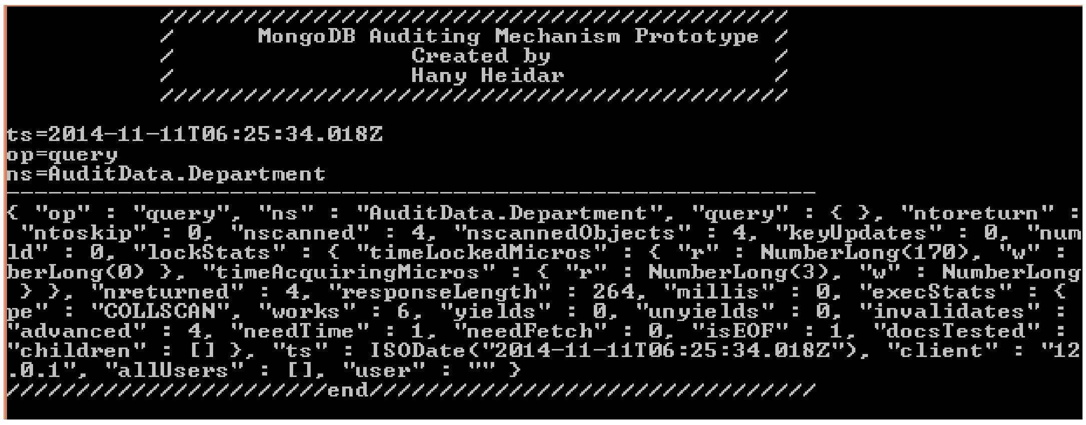
4.1 Introduction

In the way to evaluate the proposed auditing mechanism prototype, the experiment has been conducted. This experiment has been applied on two different data sets to test the auditing process of the four CRUD/DML operations in MongoDB.

Sequentially, The experiment objectives are; examine the types of operations that can be audited (Insert – Select – Update – Delete) as well as, evaluate the performance before and after applying the proposed auditing mechanism by calculating the time taken for executing CRUD/DML operations (Li & Manoharan, 2013). This chapter explores the auditing mechanism prototype, the experiment results and evaluation.

4.2 Auditing Mechanism Prototype

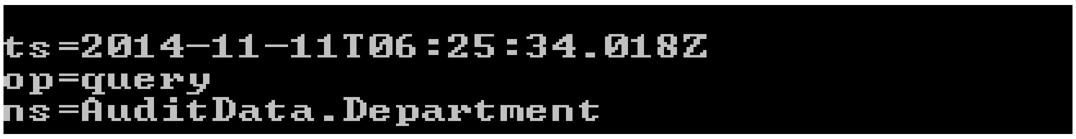
The auditing prototype was built depended on the algorithm that has been created in section 3.2.2 at the stage two of the research methodology. This prototype has been developed by using C# programming language and Microsoft Visual Studio Framework 2013. The experiment has been executed on Windows 8.1 operating system, Pentium processors Core i5, and 8GB RAM. Figure 4.1 bellow shows the auditing mechanism prototype output.



```
          /////////////////////////////////////////////////
          /       MongoDB Auditing Mechanism Prototype /
          /                  Created by               /
          /                  Hany Heidar              /
          /////////////////////////////////////////////////

ts=2014-11-11T06:25:34.018Z
op=query
ns=AuditData.Department
--------------------------------------------------------------
{ "op" : "query", "ns" : "AuditData.Department", "query" : { }, "ntoreturn" :
 "ntoskip" : 0, "nscanned" : 4, "nscannedObjects" : 4, "keyUpdates" : 0, "num
ld" : 0, "lockStats" : { "timeLockedMicros" : { "r" : NumberLong(170), "w" :
berLong(0) }, "timeAcquiringMicros" : { "r" : NumberLong(3), "w" : NumberLong
 } }, "nreturned" : 4, "responseLength" : 264, "millis" : 0, "execStats" : {
pe" : "COLLSCAN", "works" : 6, "yields" : 0, "unyields" : 0, "invalidates" :
"advanced" : 4, "needTime" : 1, "needFetch" : 0, "isEOF" : 1, "docsTested" :
"children" : [] }, "ts" : ISODate("2014-11-11T06:25:34.018Z"), "client" : "12
.0.1", "allUsers" : [], "user" : "" }
////////////////////////end/////////////////////////////////
```

Figure 4.1: Auditing Mechanism Prototype

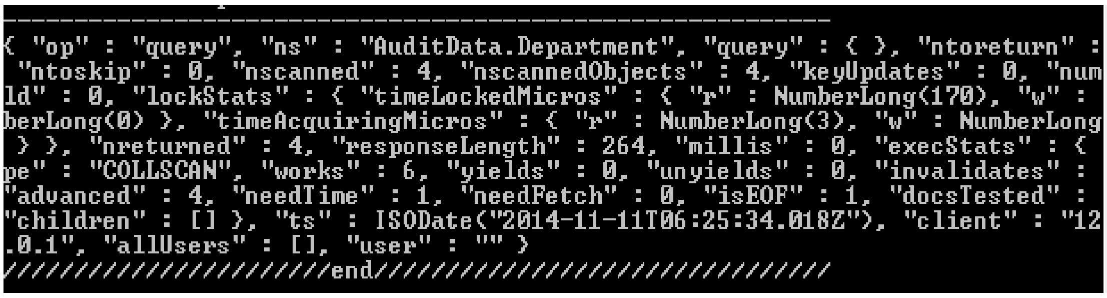
In addition, the output of the proposed mechanism prototype includes the following fields; “ts” which means time stamp that includes date and time. Also, “op” which means DML operations (query for select – remove for delete – update - insert). As well as, “ns” which means name space and denotes the target data set. All these fields are illustrated below in Figure 4.2.



```
ts=2014-11-11T06:25:34.018Z
op=query
ns=AuditData.Department
```

Figure 4.2 Fields of prototype records

Likewise, Figure 4.3 presents details field in the output of the proposed mechanism prototype. This field has detailed information about the recorded CRUD/DML operations such as number of documents retrieved and time consumed in millisecond.

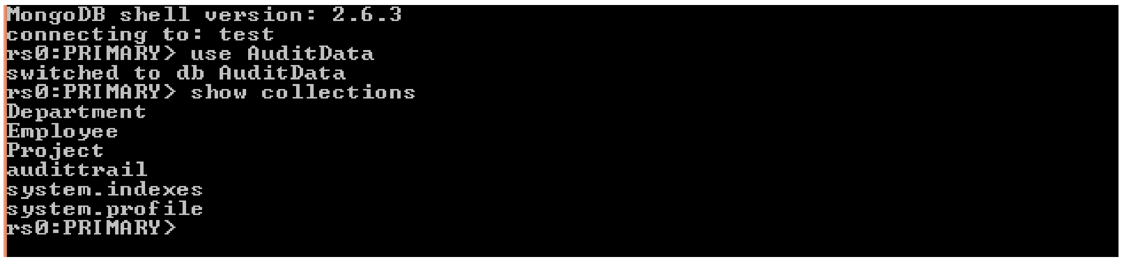


```
--------------------------------------------------------------
{ "op" : "query", "ns" : "AuditData.Department", "query" : { }, "ntoreturn" :
 "ntoskip" : 0, "nscanned" : 4, "nscannedObjects" : 4, "keyUpdates" : 0, "num
ld" : 0, "lockStats" : { "timeLockedMicros" : { "r" : NumberLong(170), "w" :
berLong(0) }, "timeAcquiringMicros" : { "r" : NumberLong(3), "w" : NumberLong
 } }, "nreturned" : 4, "responseLength" : 264, "millis" : 0, "execStats" : {
pe" : "COLLSCAN", "works" : 6, "yields" : 0, "unyields" : 0, "invalidates" :
"advanced" : 4, "needTime" : 1, "needFetch" : 0, "isEOF" : 1, "docsTested" :
"children" : [] }, "ts" : ISODate("2014-11-11T06:25:34.018Z"), "client" : "12
.0.1", "allUsers" : [], "user" : "" }
////////////////////////end/////////////////////////////////
```

Figure 4.3 *Details field of the prototype output*

4.3 The Experiment Results

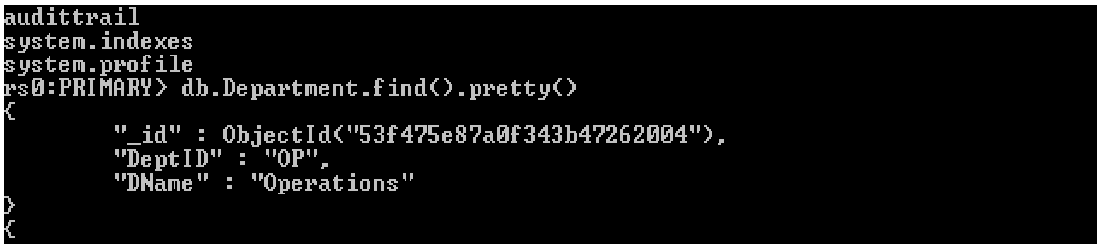
The experiment has been conducted using two different datasets which named AuditData and AuditData2. Each dataset has three data collections (tables). The first data set is “AuditData” which contains three collections; Department, Employee and Project. Figure 4.4 demonstrates the description of the first dataset “ AuditData”.



```
MongoDB shell version: 2.6.3
connecting to: test
rs0:PRIMARY> use AuditData
switched to db AuditData
rs0:PRIMARY> show collections
Department
Employee
Project
audittrail
system.indexes
system.profile
rs0:PRIMARY>
```

Figure 4.4 Data Set 1 “AuditData”

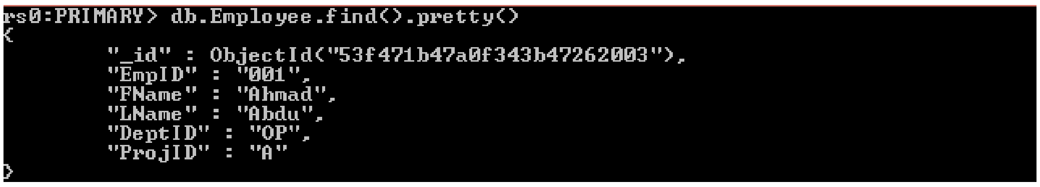
As a consequence, the Department collection has three fields for each document; which are ID, DeptID; which refers to the unique code of every department, and DName, which refers to the name of department. Figure 4.5 shows sample of data from “Department”.



```
audittrail
system.indexes
system.profile
rs0:PRIMARY> db.Department.find().pretty()
{
        "_id" : ObjectId("53f475e87a0f343b47262004"),
        "DeptID" : "OP",
        "DName" : "Operations"
}
{
```

Figure 4.5 Sample of data in “Department” collection in data set one “AuditData”

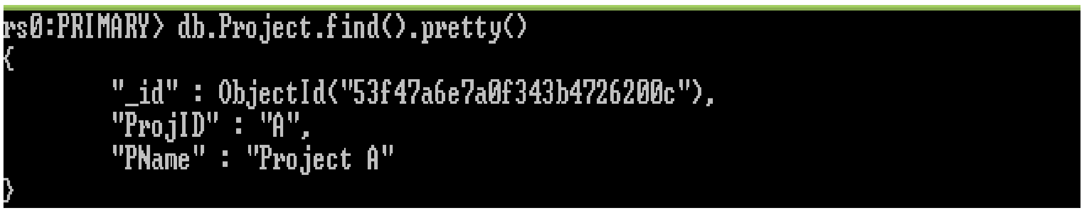
As well, Employee collection has five fields for each document; which are EmpID, FName, LName, DeptID and ProjID. Figure 4.6 provides sample of data from "Employee" collection.



```
rs0:PRIMARY> db.Employee.find().pretty()
{
        "_id" : ObjectId("53f471b47a0f343b47262003"),
        "EmpID" : "001",
        "FName" : "Ahmad",
        "LName" : "Abdu",
        "DeptID" : "OP",
        "ProjID" : "A"
}
```

Figure 4.6 Sample of data in "Employee" collection in data set one "AuditData"

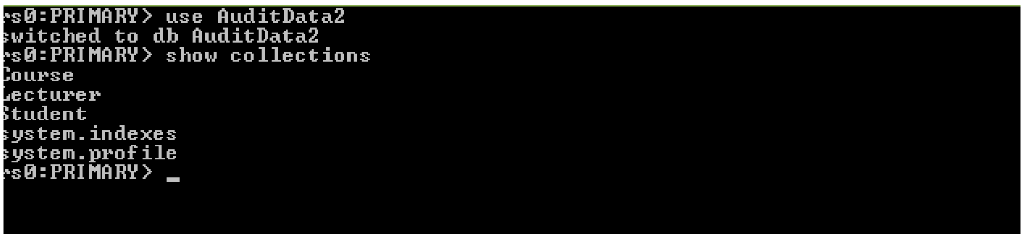
Lastly, the Project collection which has the project details. Each document in this collection has three fields which are; id, ProjID and PName. Figure 4.7 displays sample data from "Project" collection.



```
rs0:PRIMARY> db.Project.find().pretty()
{
        "_id" : ObjectId("53f47a6e7a0f343b4726200c"),
        "ProjID" : "A",
        "PName" : "Project A"
}
```

Figure 4.7 Sample of data in "Project" collection in data set one "AuditData"

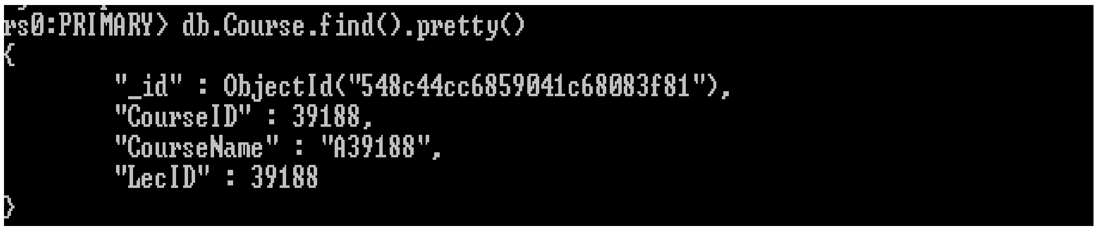
On other hand, the second dataset which is AuditData2. It consists from three collections which are; Lecturer, Course and Student. Figure 4.8 presents the description of the second dataset "AuditData2".



```
rs0:PRIMARY> use AuditData2
switched to db AuditData2
rs0:PRIMARY> show collections
Course
Lecturer
Student
system.indexes
system.profile
rs0:PRIMARY> _
```

Figure 4.8 Dataset 2 "AuditData2" description.

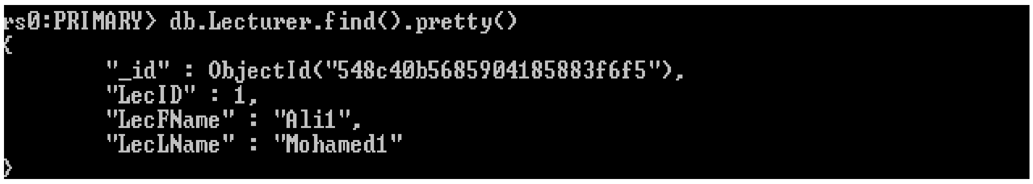
Firstly, the Course collection has four fields for each document; which are _id, CourseID, CourseName, and LecID. Figure 4.9 explains sample data from “Course” collection in data set two “AuditData2”.



```
rs0:PRIMARY> db.Course.find().pretty()
{
        "_id" : ObjectId("548c44cc6859041c68083f81"),
        "CourseID" : 39188,
        "CourseName" : "A39188",
        "LecID" : 39188
}
```

Figure 4.9 Sample of data in “Course” collection in data set two “AuditData2”

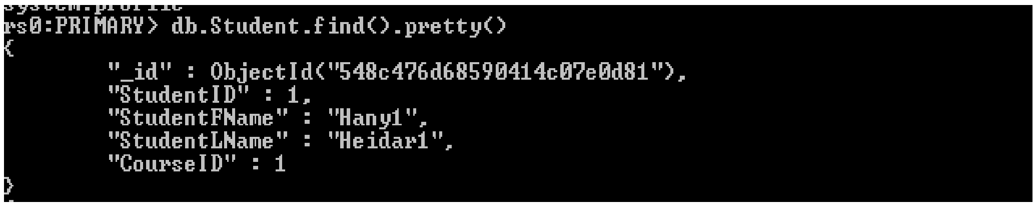
Secondly, the Lecturer collection which has the details of lecturers, consists of four fields for each document which are; _id, LecID, LecFName and LecLName. Figure 4.10 displays sample data from “Lecturer” collection.



```
rs0:PRIMARY> db.Lecturer.find().pretty()
{
        "_id" : ObjectId("548c40b5685904185883f6f5"),
        "LecID" : 1,
        "LecFName" : "Ali1",
        "LecLName" : "Mohamed1"
}
```

Figure 4.10 Sample of data in “Lecturer” collection in data set two “AuditData2”

Finally, the Student collection in second dataset “AuditData2”. Each document in “Student” collection has five fields which are; _id, StudentID, StudentFName, StudentLName, CourseID. Figure 4.11 provides sample data of Student collection.



```
system.profile
rs0:PRIMARY> db.Student.find().pretty()
{
        "_id" : ObjectId("548c476d68590414c07e0d81"),
        "StudentID" : 1,
        "StudentFName" : "Hany1",
        "StudentLName" : "Heidar1",
        "CourseID" : 1
}
```

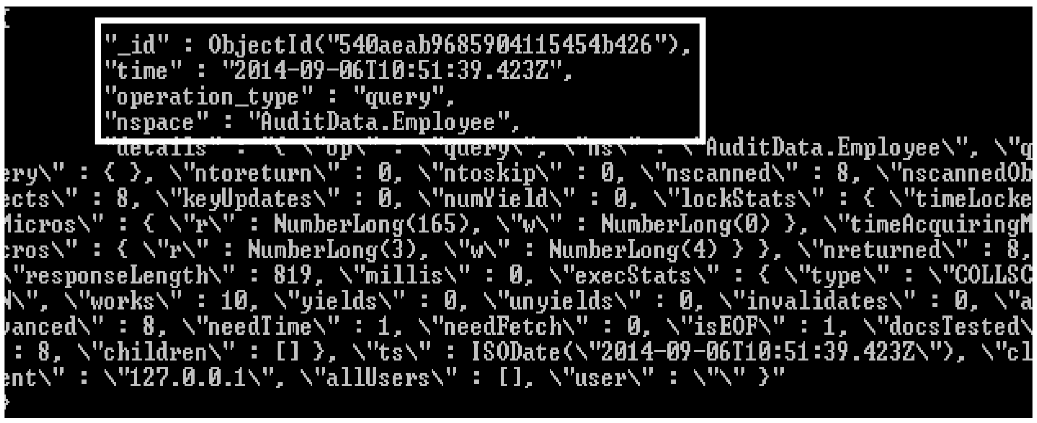
Figure 4.11 Sample of data in “Student” collection in data set two “AuditData2”

4.3.1 Results of Auditing the CRUD/DML Operations

The results of the experiment has shown that the proposed auditing mechanism prototype succeeded in auditing the four DML/CRUD operations for both of used datasets.

1. Results of First Dataset "AuditData"

The value of "operation-type" field in Figure 4.12 illustrates the type of DML/CRUD operation is a query or select operation. Similarly, the value of "time" field provides the auditing time and "nspace" field means the auditing collection.

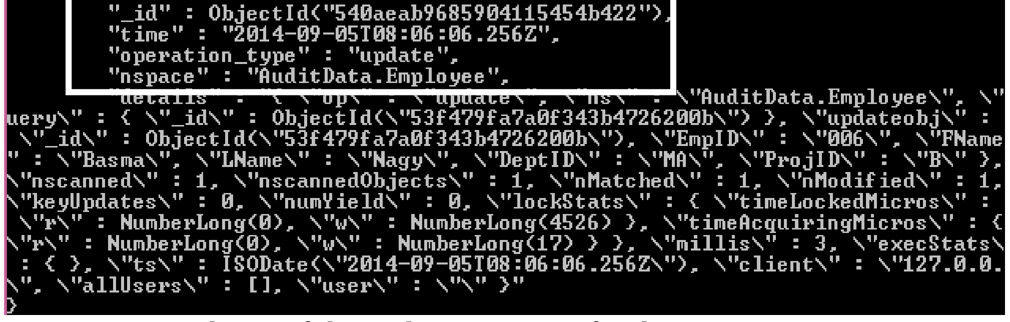


```
{
        "_id" : ObjectId("540aeab9685904115454b426"),
        "time" : "2014-09-06T10:51:39.423Z",
        "operation_type" : "query",
        "nspace" : "AuditData.Employee",
        "details" : "{ \"op\" : \"query\", \"ns\" : \"AuditData.Employee\", \"q
ery\" : { }, \"ntoreturn\" : 0, \"ntoskip\" : 0, \"nscanned\" : 8, \"nscannedOb
ects\" : 8, \"keyUpdates\" : 0, \"numYield\" : 0, \"lockStats\" : { \"timeLocke
icros\" : { \"r\" : NumberLong(165), \"w\" : NumberLong(0) }, \"timeAcquiringM
ros\" : { \"r\" : NumberLong(3), \"w\" : NumberLong(4) } }, \"nreturned\" : 8,
\"responseLength\" : 819, \"millis\" : 0, \"execStats\" : { \"type\" : \"COLLSC
N\", \"works\" : 10, \"yields\" : 0, \"unyields\" : 0, \"invalidates\" : 0, \"a
vanced\" : 8, \"needTime\" : 1, \"needFetch\" : 0, \"isEOF\" : 1, \"docsTested\
 : 8, \"children\" : [] }, \"ts\" : ISODate(\"2014-09-06T10:51:39.423Z\"), \"cl
ent\" : \"127.0.0.1\", \"allUsers\" : [], \"user\" : \"\" }"
}
```

Figure 4.12 Auditing of the query (select) operation for data set 1.

Likewise, Figure 4.13 demonstrates the update operation for Employee collection.

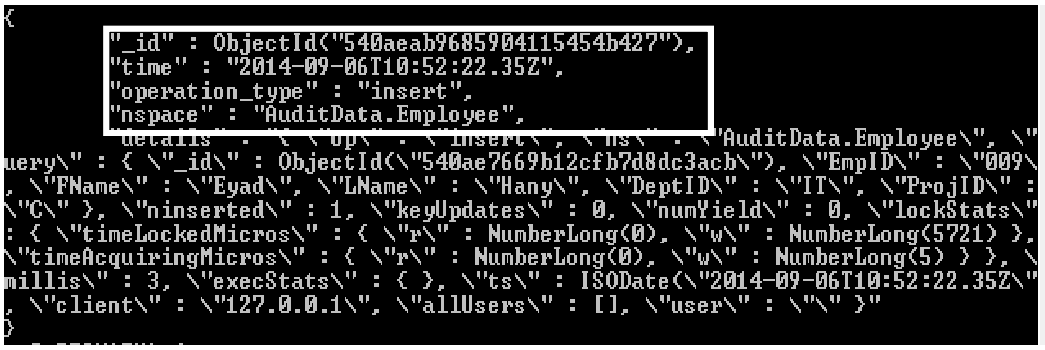
In the figure below the value of "operation-type" field is "update".



```
{
        "_id" : ObjectId("540aeab9685904115454b422"),
        "time" : "2014-09-05T08:06:06.256Z",
        "operation_type" : "update",
        "nspace" : "AuditData.Employee",
        "details" : "{ \"op\" : \"update\", \"ns\" : \"AuditData.Employee\", \"
uery\" : { \"_id\" : ObjectId(\"53f479fa7a0f343b4726200b\") }, \"updateobj\" :
 \"_id\" : ObjectId(\"53f479fa7a0f343b4726200b\"), \"EmpID\" : \"006\", \"FName
" : \"Basma\", \"LName\" : \"Nagy\", \"DeptID\" : \"MA\", \"ProjID\" : \"B\" },
\"nscanned\" : 1, \"nscannedObjects\" : 1, \"nMatched\" : 1, \"nModified\" : 1,
\"keyUpdates\" : 0, \"numYield\" : 0, \"lockStats\" : { \"timeLockedMicros\" :
 \"r\" : NumberLong(0), \"w\" : NumberLong(4526) }, \"timeAcquiringMicros\" : {
\"r\" : NumberLong(0), \"w\" : NumberLong(17) } }, \"millis\" : 3, \"execStats\
 : { }, \"ts\" : ISODate(\"2014-09-05T08:06:06.256Z\"), \"client\" : \"127.0.0.
\", \"allUsers\" : [], \"user\" : \"\" }"
}
```

Figure 4.13 Auditing of the update operation for data set 1.

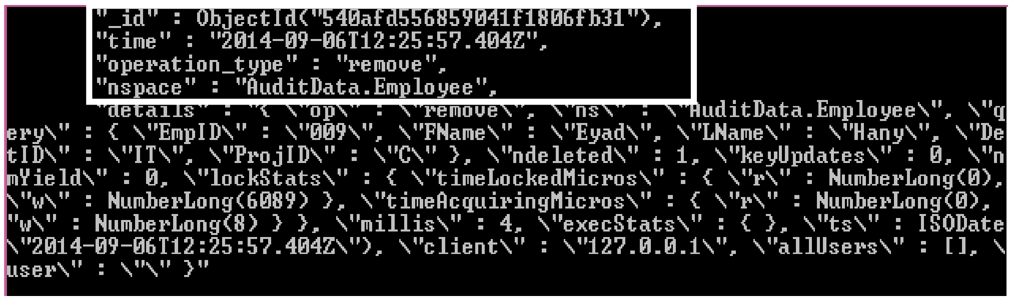
Besides the operation type, also the field “time” field provides the auditing time and “nspace” field means the auditing collection.



```
{
        "_id" : ObjectId("540aeab9685904115454b427"),
        "time" : "2014-09-06T10:52:22.35Z",
        "operation_type" : "insert",
        "nspace" : "AuditData.Employee",
        "details" : "{ \"op\" : \"insert\", \"ns\" : \"AuditData.Employee\", \"
uery\" : { \"_id\" : ObjectId(\"540ae7669b12cfb7d8dc3acb\"), \"EmpID\" : \"009\
, \"FName\" : \"Eyad\", \"LName\" : \"Hany\", \"DeptID\" : \"IT\", \"ProjID\" :
\"C\" }, \"ninserted\" : 1, \"keyUpdates\" : 0, \"numYield\" : 0, \"lockStats\"
: { \"timeLockedMicros\" : { \"r\" : NumberLong(0), \"w\" : NumberLong(5721) },
\"timeAcquiringMicros\" : { \"r\" : NumberLong(0), \"w\" : NumberLong(5) } }, \
millis\" : 3, \"execStats\" : { }, \"ts\" : ISODate(\"2014-09-06T10:52:22.35Z\"
, \"client\" : \"127.0.0.1\", \"allUsers\" : [], \"user\" : \"\" }"
}
```

Figure 4.14 Auditing of the insert operation for data set 1.

The value of “operation-type” field in Figure 4.14 describes the type of DML/CRUD operation is an insert operation. Similarly, the value of “time” field provides the auditing time and “nspace” field means the auditing collection.

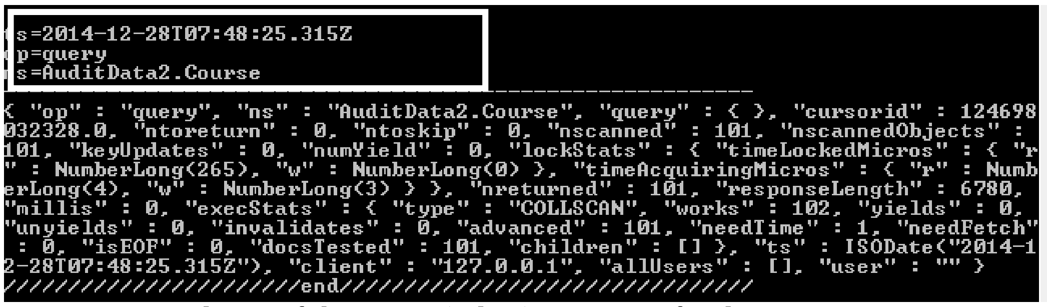


```
        "_id" : ObjectId("540afd556859041f1806fb31"),
        "time" : "2014-09-06T12:25:57.404Z",
        "operation_type" : "remove",
        "nspace" : "AuditData.Employee",
        "details" : "{ \"op\" : \"remove\", \"ns\" : \"AuditData.Employee\", \"q
ery\" : { \"EmpID\" : \"009\", \"FName\" : \"Eyad\", \"LName\" : \"Hany\", \"De
tID\" : \"IT\", \"ProjID\" : \"C\" }, \"ndeleted\" : 1, \"keyUpdates\" : 0, \"n
mYield\" : 0, \"lockStats\" : { \"timeLockedMicros\" : { \"r\" : NumberLong(0),
\"w\" : NumberLong(6089) }, \"timeAcquiringMicros\" : { \"r\" : NumberLong(0),
"w\" : NumberLong(8) } }, \"millis\" : 4, \"execStats\" : { }, \"ts\" : ISODate
\"2014-09-06T12:25:57.404Z\"), \"client\" : \"127.0.0.1\", \"allUsers\" : [], \
user\" : \"\" }"
```

Figure 4.15 Auditing of the remove (delete) operation for data set 1.

Similarly, as Figure 4.15 demonstrates the operation type is “remove” and “nspace” is Employee collection.

2. Results of Second Data Set “AuditData2”



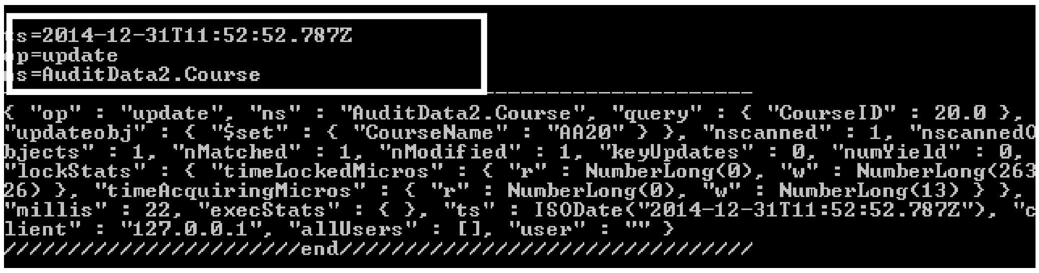
```
s=2014-12-28T07:48:25.315Z
p=query
s=AuditData2.Course
-------------------------------------------------------------------------
{ "op" : "query", "ns" : "AuditData2.Course", "query" : { }, "cursorid" : 124698
032328.0, "ntoreturn" : 0, "ntoskip" : 0, "nscanned" : 101, "nscannedObjects" :
101, "keyUpdates" : 0, "numYield" : 0, "lockStats" : { "timeLockedMicros" : { "r
" : NumberLong(265), "w" : NumberLong(0) }, "timeAcquiringMicros" : { "r" : Numb
erLong(4), "w" : NumberLong(3) } }, "nreturned" : 101, "responseLength" : 6780,
"millis" : 0, "execStats" : { "type" : "COLLSCAN", "works" : 102, "yields" : 0,
"unyields" : 0, "invalidates" : 0, "advanced" : 101, "needTime" : 1, "needFetch"
 : 0, "isEOF" : 0, "docsTested" : 101, "children" : [] }, "ts" : ISODate("2014-1
2-28T07:48:25.315Z"), "client" : "127.0.0.1", "allUsers" : [], "user" : "" }
///////////////////////////end/////////////////////////////////////
```

Figure 4.16 Auditing of the query (select) operation for data set 2.

The field “operation-type” in Figure 4.16 has “query” value which means that the type of DML/CRUD operation that has been audited is the query or select operation.

Likewise, Figure 4.17 demonstrates the update operation for Course collection.

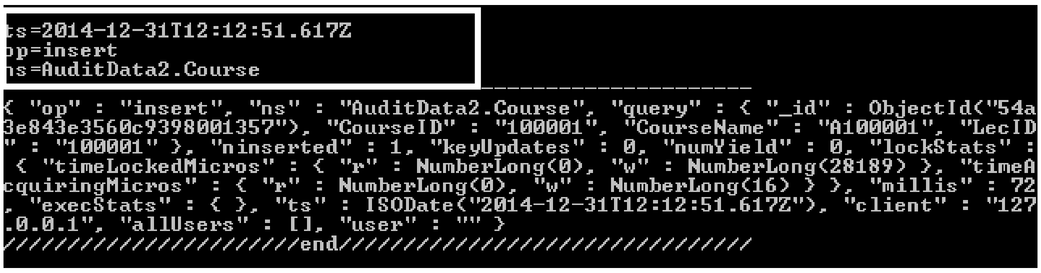
In the figure below the value of “operation-type” field is “update”.



```
s=2014-12-31T11:52:52.787Z
p=update
s=AuditData2.Course
-------------------------------------------------------------------------
{ "op" : "update", "ns" : "AuditData2.Course", "query" : { "CourseID" : 20.0 },
"updateobj" : { "$set" : { "CourseName" : "AA20" } }, "nscanned" : 1, "nscannedO
bjects" : 1, "nMatched" : 1, "nModified" : 1, "keyUpdates" : 0, "numYield" : 0,
"lockStats" : { "timeLockedMicros" : { "r" : NumberLong(0), "w" : NumberLong(263
26) }, "timeAcquiringMicros" : { "r" : NumberLong(0), "w" : NumberLong(13) } },
"millis" : 22, "execStats" : { }, "ts" : ISODate("2014-12-31T11:52:52.787Z"), "c
lient" : "127.0.0.1", "allUsers" : [], "user" : "" }
///////////////////////////end/////////////////////////////////////
```

Figure 4.17 Auditing of the update operation for data set 2.

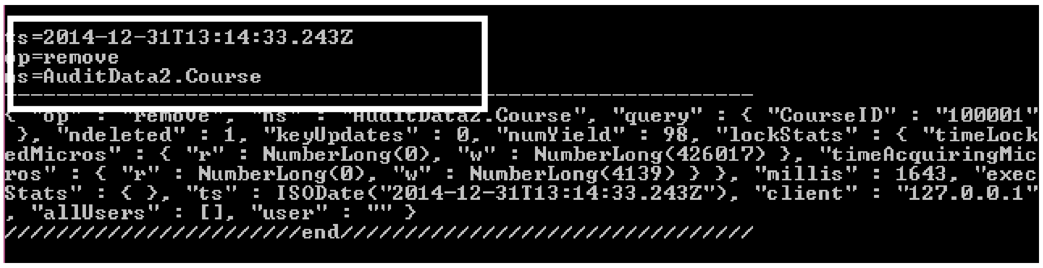
Besides the operation type, also the field “time” field provides the auditing time and “nspace” field means the auditing collection.



```
ts=2014-12-31T12:12:51.617Z
op=insert
ns=AuditData2.Course
-------------------------------------------------------------------------
{ "op" : "insert", "ns" : "AuditData2.Course", "query" : { "_id" : ObjectId("54a
3e843e3560c9398001357"), "CourseID" : "100001", "CourseName" : "A100001", "LecID
" : "100001" }, "ninserted" : 1, "keyUpdates" : 0, "numYield" : 0, "lockStats" :
 { "timeLockedMicros" : { "r" : NumberLong(0), "w" : NumberLong(28189) }, "timeA
cquiringMicros" : { "r" : NumberLong(0), "w" : NumberLong(16) } }, "millis" : 72
, "execStats" : { }, "ts" : ISODate("2014-12-31T12:12:51.617Z"), "client" : "127
.0.0.1", "allUsers" : [], "user" : "" }
///////////////////////////end/////////////////////////////////////
```

Figure 4.18 Auditing of the insert operation for data set 2.

The field “operation-type” in Figure 4.18 depicts the type of DML/CRUD operation that has been audited and the operation here is the insert operation.



```
ts=2014-12-31T13:14:33.243Z
op=remove
ns=AuditData2.Course
------------------------------------------------------------
{ "op" : "remove", "ns" : "AuditData2.Course", "query" : { "CourseID" : "100001"
 }, "ndeleted" : 1, "keyUpdates" : 0, "numYield" : 98, "lockStats" : { "timeLock
edMicros" : { "r" : NumberLong(0), "w" : NumberLong(426017) }, "timeAcquiringMic
ros" : { "r" : NumberLong(0), "w" : NumberLong(4139) } }, "millis" : 1643, "exec
Stats" : { }, "ts" : ISODate("2014-12-31T13:14:33.243Z"), "client" : "127.0.0.1"
, "allUsers" : [], "user" : "" }
///////////////////////////end///////////////////////////////////////
```

Figure 4.19 Auditing of the remove operation for data set 2.

The field “operation-type” in Figure 4.19 depicts the type of DML/CRUD operation that has been audited and the operation here is the remove operation.

In summary, the aforementioned results explicated that the proposed auditing mechanism succeeded in auditing the four CRUD/DML operations. It has been achieved by implementing the auditing mechanism on two data sets AuditData and AuditData2. Accordingly, the auditing mechanism recorded the information of audited CRUD/DML operation in an audit trail. These information provide the time, operation type, name of document and the details. As well, the auditing mechanism stored these information in an audit trail in the same MongoDB database.

4.4 Performance Evaluation

The second objective of the experiment measures the impact of the proposed auditing mechanism on the performance of MongoDB database. In order to measure and evaluate the performance, the technique conducted by Boicea et al. (2012) and also Li and Manoharan (2013) has been followed. Continually, measuring the performance has achieved by calculating the time taken for executing CRUD/DML operations before and after applying the proposed mechanism.

Details of experiment results and evaluation are demonstrated in the following sections with tables and charts. Furthermore, in the incoming tables, the number of operations refers to the number of execution times for a certain operation. As well, the tables illustrate the time taken in millisecond. By the same way, the charts explicate the time taken in millisecond on the vertical axis and number of execution times for an operation on the horizontal axis.

.

Table 4.1

Time of the select operations in both data sets MS)

Data Set		**Number of Operations**				
		10	**50**	**100**	**1000**	**10000**
Before	**Data Set 1**	156	162	165	203	441
	Data Set 2	142	145	148	178	375
After	**Data Set 1**	413	737	1099	7254	18132
	Data Set 2	1838	7901	15393	150460	1529703

Table 4.1 has presented the execution time in millisecond of the select operation in both data sets AuditData and AuditData2. The time has been calculated before and after implementing the proposed auditing mechanism. Also, the number of operations refers to the number of execution times for the select operation.

The comparison in Table 4.1 above showed a difference between before and after applying the auditing mechanism in the both datasets. This difference increasing gradually and it reached the maximum in case of 10000 number of operations. Accordingly, this difference is considered inadequate because the increase in time taken means that the proposed auditing mechanism negatively impacted MongoDB performance by decreasing the performance of database.

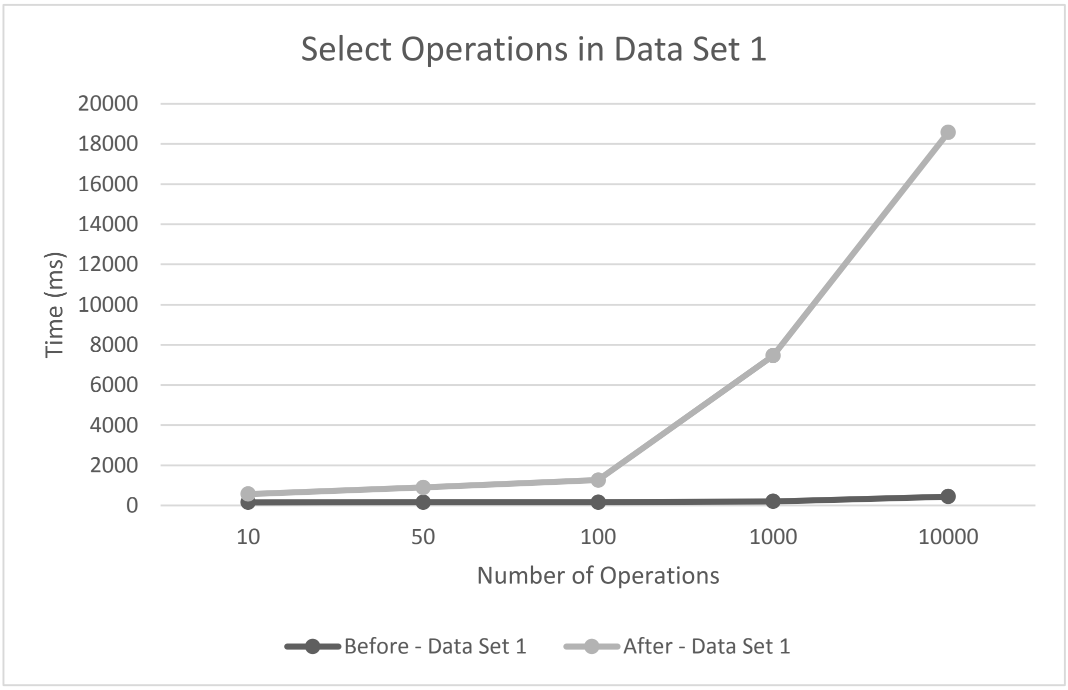


Figure 4.20 Time of the select operation for Data Set 1 before and after applying the proposed auditing mechanism

The chart in Figure 4.20 showed the time taken in millisecond on the vertical axis and number of times the select operation is executed on the horizontal axis in case of data set 1. Furthermore, Figure 4.20 depicted a small difference in time taken in case of the number of operations was 10, 50 or 100 which is considered good because the impact on the performance of database is very limited. Moreover, the difference in time significantly increased in case of the number of operations was 1000 and 10000.

Accordingly, this significant difference is considered incompetent because it means that the proposed auditing mechanism decreased the performance of MongoDB database.

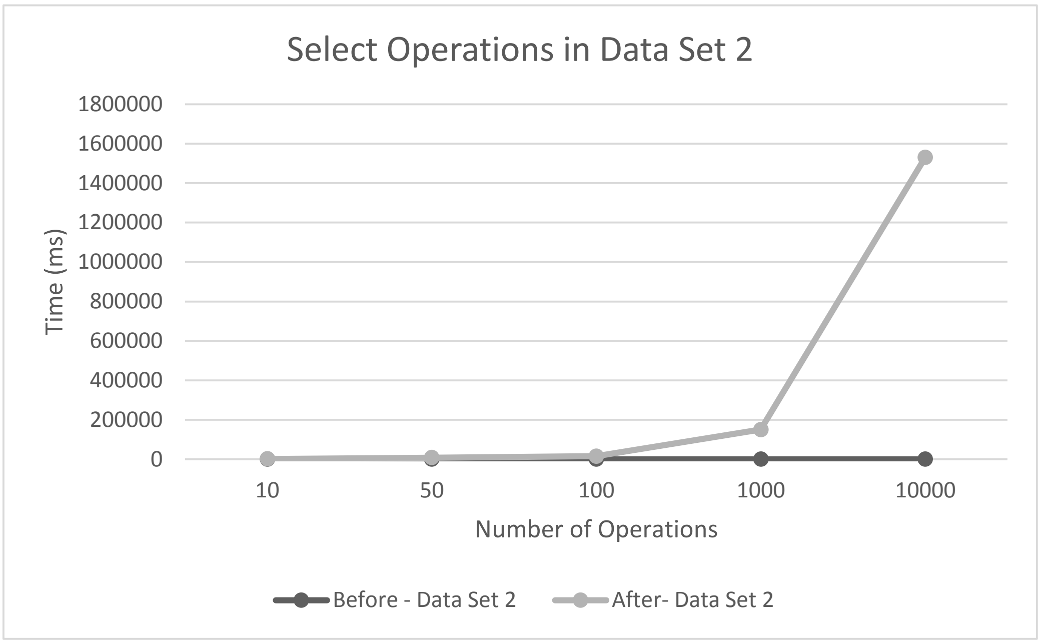


Figure 4.21 Time of the select operation for Data Set 2 before and after applying the proposed auditing mechanism.

The line chart in Figure 4.21 presents the time taken in millisecond on the vertical axis and number of times the select operation is executed on the horizontal axis in case of data set 2. Furthermore, the chat illustrates almost a little unseen difference in time taken in case the number of operations was 10, 50 or 100 which is considered good because the performance of database is almost similar in the both cases before and after applying the mechanism. Moreover, the difference in time increased in case of the number of operations was 1000 and 10000. Accordingly, this difference is considered inadequate because that the proposed auditing mechanism reduced the performance of MongoDB database.

Table 4.2

Time of the insert operations in both data sets (MS)

Data Set		Number of Operations				
		10	**50**	**100**	**1000**	**10000**
Before	**Data Set 1**	439	482	520	1272	8126
	Data Set 2	402	427	483	1178	7991
After	**Data Set 1**	563	660	952	6210	25382
	Data Set 2	544	788	861	5264	19760

Table 4.2 presented the execution time in millisecond of the insert operation in both datasets AuditData and AuditData2 before and after applying the proposed auditing mechanism. As it is cleared from the comparison that the difference between the two cases increased in a gradual way until it arrived the maximum in case of 10000 number of operations. The increase in time taken demonstrates the negative impact of the proposed auditing mechanism on the performance of MongoDB database.

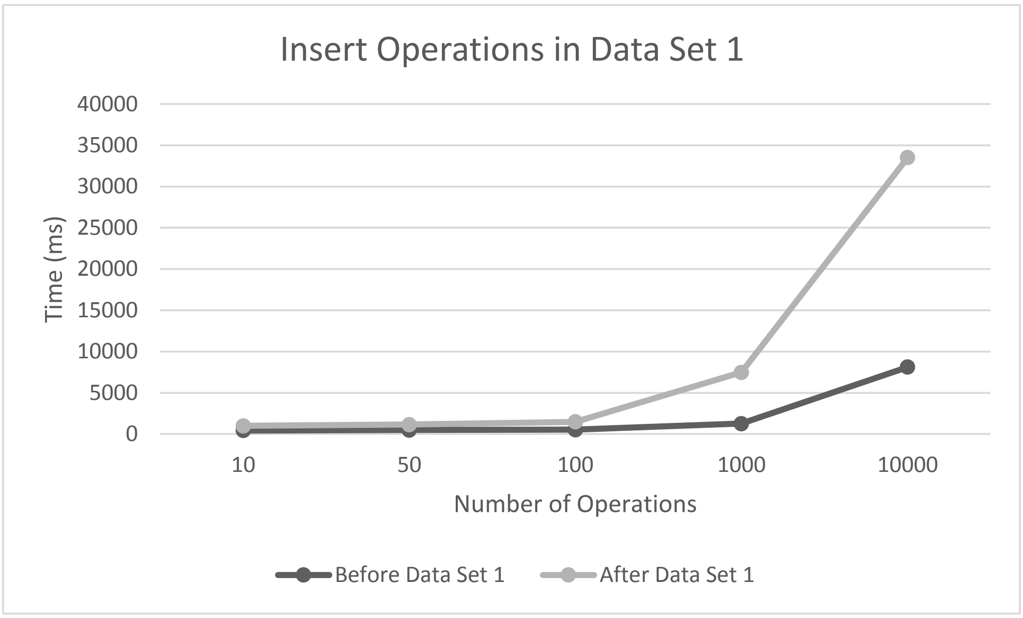


Figure 4.22 Time of the insert operation for Data Set 1 before and after applying the proposed Auditing mechanism.

The chart in Figure 4.22 showed the time taken in millisecond on the vertical axis and number of insert operations on the horizontal axis in case of data set 1. Furthermore, Figure 4.22 depicted there is almost no difference in time taken in case of the number of operations was 10, 50 or 100 which is considered good because the performance of database is very close in the both cases before and after applying the mechanism. Moreover, the difference in time increased in case of the number of operations was 1000 and 10000. Accordingly, this difference means that the proposed auditing mechanism declined the performance of MongoDB database.

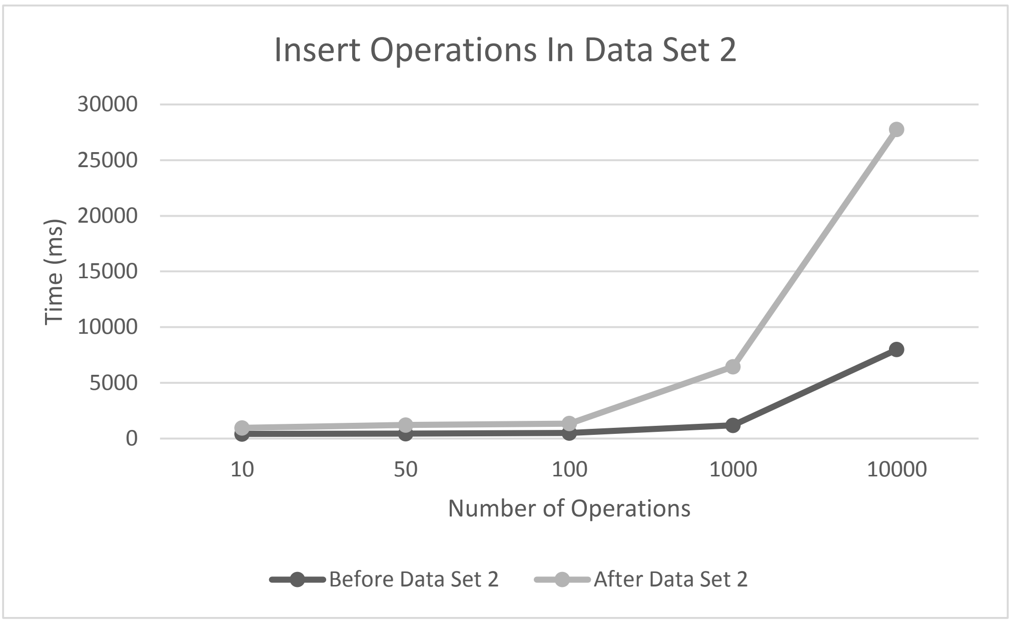


Figure 4.23 Time of the insert operation for Data Set 2 before and after applying the proposed Auditing mechanism.

The chart in Figure 4.23 showed the time taken in millisecond on the vertical axis and number of times the insert operation is executed on the horizontal axis in case of data set 2. Furthermore, the chart depicted a little difference in time taken in case of the number of operations was 10, 50 or 100 which means that the impact on the performance of database is very limited and this is considered good. Furthermore, the difference in time significantly increased in case of the number of operations was 1000 and 10000. Accordingly, this significant difference is considered inadequate because it means that the proposed auditing mechanism reduced the performance of MongoDB database.

Table 4.3

Time of the remove/delete operations in both data sets (MS)

Data Set		Number of Operations				
		10	**50**	**100**	**1000**	**10000**
Before	**Data Set 1**	1157	4023	7702	72138	758629
	Data Set 2	1168	4129	7588	77064	754924
After	**Data Set 1**	1833	7100	14322	123240	1088074
	Data Set 2	1844	7205	13638	130231	1078074

Table 4.3 presented the average time in millisecond of the remove/delete operation in both data sets AuditData and AuditData2 before and after implementing the proposed auditing mechanism. In the table, the number of operations refers to the number of times the remove/delete operation is executed in the test.

As the data showed from the comparison that the difference between the two cases increased gradually until it arrived at the maximum in case of 10000 number of operations. The increase in time taken generally is considered incompetent because it means that the proposed auditing mechanism negatively impacted MongoDB performance by decreasing the performance of database.

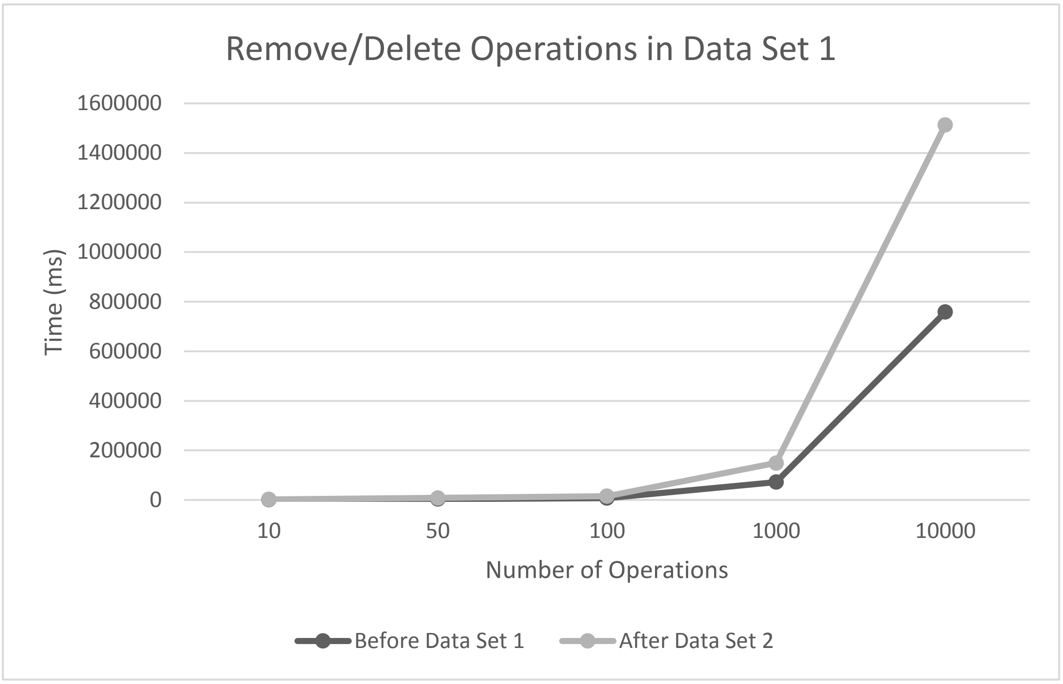


Figure 4.24 Time of the remove/delete operation for Data Set 1 before and after applying the proposed auditing mechanism.

The chart in Figure 4.24 presented the time taken in millisecond on the vertical axis and number of times the remove/delete operation is executed on the horizontal axis in case of data set 1. Furthermore, Figure 4.24 depicted a small difference in time taken in case of the number of operations was 10, 50, 100 or 1000 which is considered good to some extent because the impact on the performance of database is very limited. On the other hand, the difference in time significantly increased in the case of the number of operations was 10000 which is considered incompetent because it means that the proposed auditing mechanism decreased the performance of MongoDB database.

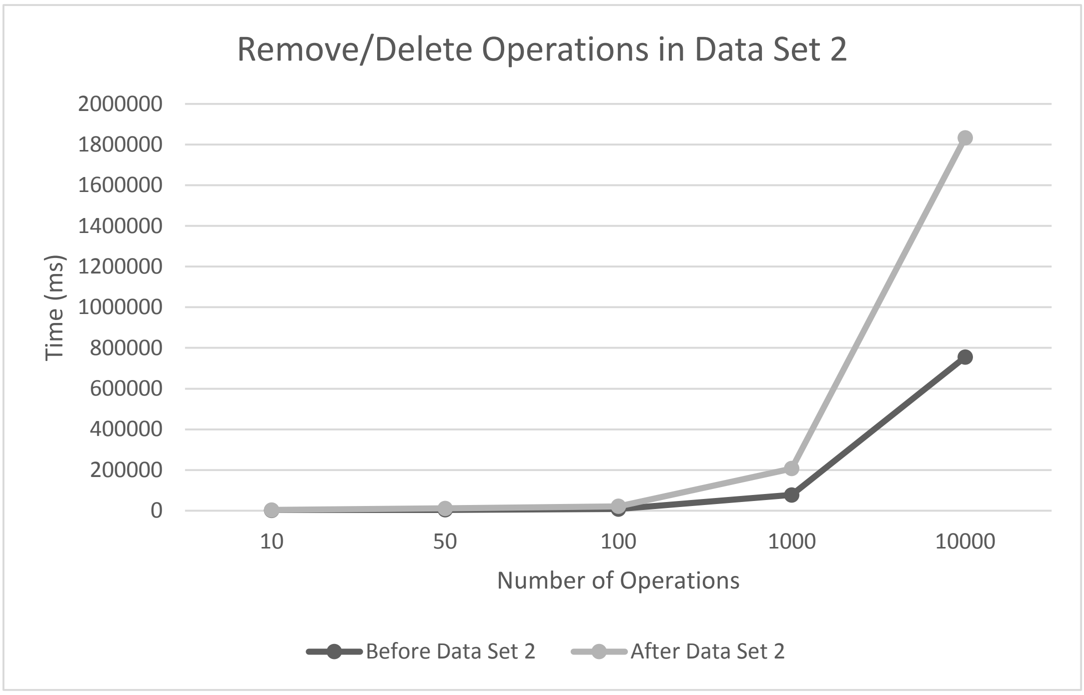


Figure 4.25 Time of the remove/delete operation for Data Set 2 before and after applying the proposed Auditing mechanism.

The chart in Figure 4.25 showed the time taken in millisecond on the vertical axis and number of times the remove/delete operation is executed on the horizontal axis in case of data set 2. Furthermore, Figure 4.25 depicted a small difference in time taken in case of the number of operations was 10, 50, 100 or 1000. This difference could be good since its impact on the performance of database is limited. Nevertheless, the difference in time significantly increased in the case of the number of operations was 10000 which is considered inadequate because it means that the proposed auditing mechanism reduced the performance of MongoDB database.

Table 4.4

Time of the update operations in both data sets (MS)

Data Set		Number of Operations				
		10	**50**	**100**	**1000**	**10000**
Before	**Data Set 1**	438	466	507	1207	8054
	Data Set 2	1111	3943	7396	70775	704924
After	**Data Set 1**	473	693	1027	6745	26067
	Data Set 2	1270	4274	7975	74470	758343

Table 4.4 presented the average time in millisecond of the update operation in both datasets (AuditData and AuditData2) before and after implementing the proposed auditing mechanism. In the table, the number of operations refers to the number of times the update operation is executed in the test. As the data showed from the comparison, the difference in execution time calculated is increasing gradually and it reached the peak where number of operations was 10000. The increase in time taken means that the proposed auditing mechanism negatively impacted MongoDB performance by decreasing the performance of database.

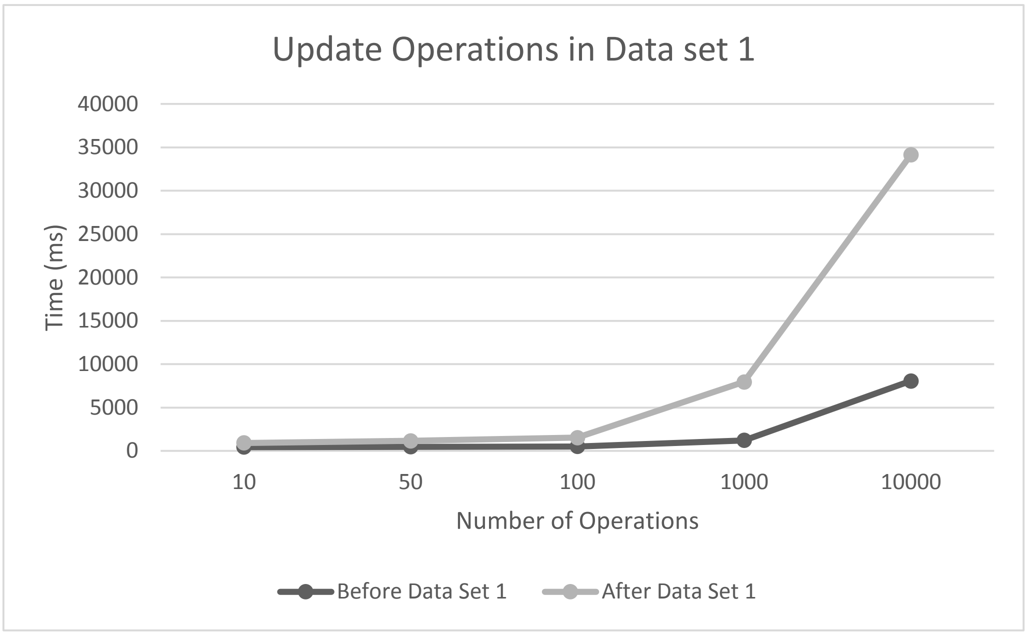


Figure 4.26 Time of the update operation in data set 1 before and after applying the proposed Auditing mechanism.

The chart in Figure 4.26 showed the time taken in millisecond on the vertical axis and number of times the update operation is executed on the horizontal axis in case of data set 1. Furthermore, the chart depicted a very small difference in time taken in case of the number of operations was 10, 50 or 100 which means that the impact on the performance of database is very limited and this is considered good. Furthermore, the difference in time significantly increased in case of the number of operations was 1000 and 10000. Sequentially, this significant difference is considered inadequate because it means that the proposed auditing mechanism decreased the performance of MongoDB database.

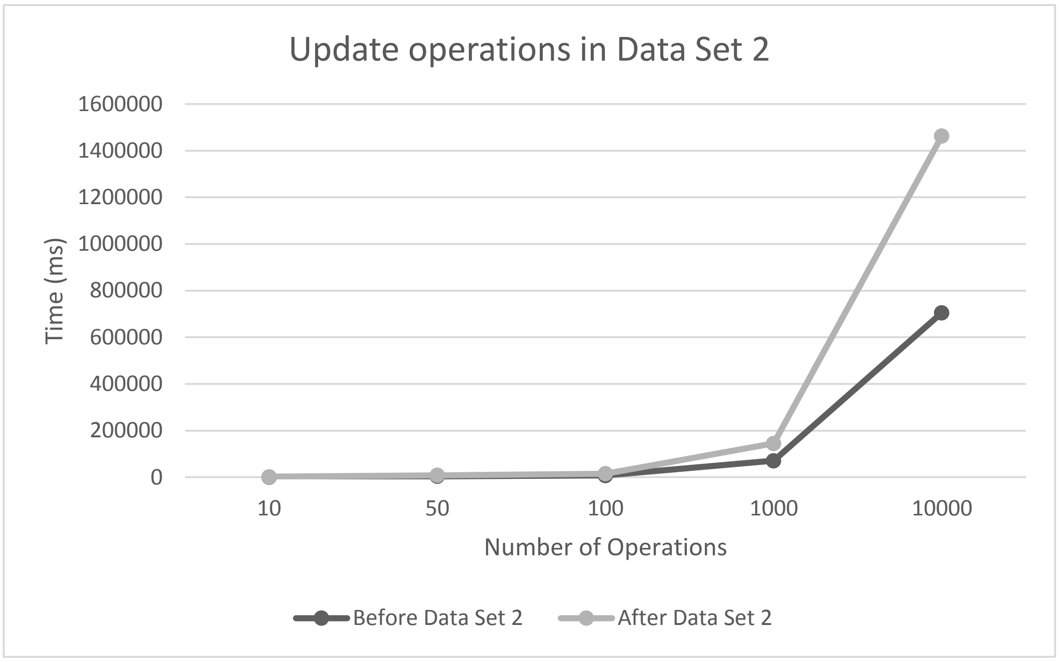


Figure 4.27 Time of the update operation in data set 2 before and after applying the proposed auditing mechanism.

The chart in Figure 4.27 showed the time taken in millisecond on the vertical axis and number of times the update operation is executed on the horizontal axis in case of data set 2. Furthermore, the chat depicted almost little unseen difference in time taken in case the number of operations was 10, 50 or 100 which is considered good because the performance of database is almost similar in the both cases before and after applying the mechanism. Moreover, the difference in time increased in case of the number of operations was 1000 and 10000. Accordingly, this difference is considered incompetent because that the proposed auditing mechanism reduced the performance of MongoDB database.

4.5 Summary

This chapter explained the auditing mechanism prototype, the experiment results and evaluation. Firstly, section 4.2 explored the prototype and programing language and the tools used to develop it in addition to the output of this prototype. Secondly, section 4.3 described the experiment results and how the new auditing mechanism successfully achieved the auditing for CRUD/DML operations. It has been accomplished by implementing the auditing mechanism on two datasets AuditData and AuditData2.

Moreover, section 4.3.1 explained the types of audited CRUD/DML operations in first and second datasets. Furthermore, this section illustrated how that the auditing mechanism recorded the information of audited CRUD/DML operation in an audit trail. These information provide the time, operation type and name of document in beside of more details. In addition, the audit trail that has the auditing information is existed in the same MongoDB database.

Eventually, section 4.4 demonstrated the impact of the new auditing mechanism on the performance of MongoDB database. In order to measure and evaluate the performance, this study followed Boicea et al. (2012) and Li and Manoharan (2013). For the sake of measuring the performance the time taken for executing CRUD/DML operations has been calculated before and after applying the proposed mechanism.

After applying this technique on two datasets AuditData and AuditData2. Generally, it is noticed that the auditing mechanism is considered competent in the case of number of CRUD/DML operations are small such as 10 and 50. On contrary, the performance of MongoDB affected negatively when the number of CRUD/DML operations were big such as 1000 and 10000 operations.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5.1 Introduction

This study has demonstrated important issues related to the auditing functionality in MongoDB, and according to what have achieved, this chapter concludes the outcomes of this study and presents limitation and future work.

5.2 Conclusion

In this study, the auditing gap in NoSQL Databases and MongoDB database were explained carefully. As stated in Chapter One, there three main objectives were formulated; 1) identifying the auditing features of MongoDB, 2) improving the auditing features in MongoDB by developing a new auditing mechanism, and finally, 3) evaluating the proposed mechanism.

In order to identify MongoDB auditing features as the first objective, the study depended on the literatures and related researches which depicted that MongoDB provides auditing features for DDL operations but unfortunately does not present auditing for CRUD/DML operations.

Based on the lack of auditing determined in the first objective, the study moved to the second objective; improving the auditing features in MongoDB by developing a new mechanism. The architecture of mechanism consists of four components MongoDB database as a target database for auditing, audit tool and its function is extracting CRUD/DML operations from target database, interface for displaying audited record audit trail as a log to store the extracted CRUD/DML operations. In order to implement the proposed mechanism, it was needed to create algorithm and prototype for this algorithm. Accordingly, to evaluate the proposed mechanism after creating the algorithm which is the third objective an experiment was conducted.

Sequentially, the experiment results shows that the proposed auditing mechanism succeeded in auditing the four CRUD/DML operations (query/select – insert – update-remove/delete). In addition, it is noticed that the proposed auditing mechanism affect the performance of MongoDB by increasing the time of executions and this increase has a positive correlation with the number of CRUD/DML operations.

In conclusion, this study tried to fill the gap represented in the lack of CRUD/DML auditing feature existing in an open source NoSQL database MongoDB. The research reached this goal through achieving the three of the study.

5.3 Limitations

First limitation was the unavailability of real data to use in this study as it was explained in section 3.2.3 in Chapter Three that experimental data sets were used to test the proposed mechanism. Second limitation was the limited resources as it was mentioned that the testing is conducted on single node but it is supposed to be conducted on multiple nodes.

5.4 Future Work

It is suggested to implement the proposed auditing mechanism in real environment to achieve real evaluation. Also conducting the evaluation in multiple nodes and multi-client environments may lead to better assessment and evaluation for the proposed mechanism. Furthermore, a friendly user interface could be built for the mechanism’s prototype.

REFERENCES

Apache CouchDB. (2014).Retrieved 12 March, 2014 from Apache CouchDB: http: //couchdb. apache.org/

Apache HBase. (2014). HBase - Apache HBase™ Home. Retrieved 12 March, 2014 from http://hbase.apache.org

Boicea, A., Radulescu, F., & Agapin, L. I. (2012). MongoDB vs Oracle-Database Comparison. *2012 Third International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)* (pp. 330-335).

Bonnet, L., Laurent, A., Sala, M., Laurent, B., & Sicard, N. (2011). Reduce, you say: What nosql can do for data aggregation and bi in large repositories. *In Database and Expert Systems Applications (DEXA), 2011 22nd International Workshop on* (pp. 483-488). IEEE.

Buerli, M., & Obispo, C. P. S. L. (2012). The Current State of Graph Databases. Retrieved 7 May, 2014 from http://www.cs.utexas.edu: http://www.cs.utexas.edu~cannata/dbms/Class%20Notes/09%20Graph_Databases_Survey.pdf

Couchbase Server the NoSQL document database. (2014). Couchbase Server Distributed, Non-Relational Database Couchbase. Retrieved from http://www.couchbase.com/couchbase-server/overview

Dean, J., & Ghemawat, S. (2010). MapReduce: a flexible data processing tool. Communications of the ACM, 53(1), 72-77.

Ezumalai, R., & Aghila, G. (2009). Combinatorial approach for preventing SQL Injection attacks. In *Advance Computing Conference, 2009. IACC 2009. IEEE International* (pp. 1212-1217). IEEE.

Geer, D. (2005). Malicious bots threaten network security. *Computer*, *38*(1), 18-20.

Dijcks, J. P. (2012). Oracle: Big data for the enterprise. Oracle White Paper.

Gantz, J., & Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the Far East. IDC iView: IDC Analyze the Future. Retrieved 12 March, 2014 from www.emc.com: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system. In ACM SIGOPS Operating Systems Review (Vol. 37, No. 5, pp. 29-43). ACM.

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, (2)*, 2-22*.

Hecht, R., & Jablonski, S. (2011). NoSQL Evaluation. *International Conference on Cloud and Service Computing*, (pp. 337-341).

Hsu, W. C., Huang, J. Y., Chen, C. H., Su, C. Y., Shih, H. C., Liao, T. Y., & Liao, I. E. (2013). A cloud service for the evaluation of company's financial health using XBRL-based financial statements. In *Big Data, 2013 IEEE International Conference on* (pp. 10-14). IEEE.

Kadebu, P., & Mapanga, I. (2014). A Security Requirements Perspective towards a Secured NOSQL Database Environment. *International Conference of Advance Research and Innovation (ICARI-2014), (3), 472-480.*

Kanade, A., Gopal, A., & Kanade, S. (2014, February). A study of normalization and embedding in MongoDB. In *Advance Computing Conference (IACC), 2014 IEEE International* (pp. 416-421). IEEE.

Kanade, A. S., Gopal, A., & Kanade, S. (2013). Cloud Based Databases-A Changing Trend. International Journal of Management, IT and Engineering, 3(7), 273-287.

Lawrence, R. (2014). Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB. In Computational Science and Computational Intelligence (CSCI), 2014 International Conference on (Vol. 1, pp. 285-290). IEEE.

Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. In Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on (pp. 15-19). IEEE.

Liang, J., & Mizuno, O. (2011). Analyzing Involvements of Reviewers Through Mining A Code Review Repository. In *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)* (pp. 126-132). IEEE.

Liu, L., & Huang, Q. (2009). A framework for database auditing. In Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on (pp. 982-986). IEEE.

Liu, Y., Wang, Y., & Jin, Y. (2012). Research on the improvement of MongoDB Auto-Sharding in cloud environment. In *Computer Science & Education (ICCSE), 2012 7th International Conference on* (pp. 851-854). IEEE.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. Retrieved from http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation.

Mapanga, I., & Kadebu, P. (2013). Database Management Systems: A NoSQL Analysis. International journal of Modern Communication Technologies and Research, 1(7), 12-18.

Mohamed, M.A., Altrafi, O.G., & Ismail, M. O. (2014). Realtional vs. NoSQL A survey. *International Journal of Computer and Information Technology*, 3(3), 589-601

MongoDB. (2014). Retrieved 1 March, 2014 from http://www.mongodb.org/

Mullins, C. S. Retrieved 9 May, 2014 from www.oowidgets.com:

http://www.oowidgets.com/Database%20Auditing%20Essentials.pdf

Murugesan, P., & Ray, I. (2014). Audit Log Management in MongoDB. In Services (SERVICES), 2014 IEEE World Congress on (pp. 53-57). IEEE.

Narde, R. (2013). A Comparison of NoSQL systems (Doctoral dissertation, Rochester Institute of Technology).

Neo4j. (2014). Neo4j - The World's Leading Graph Database. Retrieved 12 March, 2014 from http://www.neo4j.org/

Ohlhorst, F. J. (2012). Big data analytics: turning big data into big money. John Wiley & Sons.

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security issues in nosql databases. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on (pp. 541-547). IEEE.

Pavlenko, D. (2014). MongoDB Audit Logging or How to Log Data Changes Using MongoDB. Retrieved 12 March, 2014 from sysgears.com: http://sysgears.com/articles/mongodb-audit-logging-or-how-log-data-changes-using-mongodb/

PCI Security Standards Council. (2010). Payment card industry (pci) data security standard – requirements and security assessment pro-cedures version 2. 0. Wakefield, MA, USA: Author. Retrieved 17 March, 2014 from:https://www.pcisecuritystandards.org/documents/pcidss v2.pdf

Pozzani, G. (2013). Introduction to NoSQL. Retrieved 21 March, 2014 from profs.sci.univr.it:http://profs.sci.univr.it/~pozzani/Materiale/nosql/01%20-%20introduction.pdf

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2007). The end of an architectural era :(it's time for a complete rewrite). In Proceedings of the 33rd international conference on Very large data bases (pp. 1150-1160). VLDB Endowment.

Rutishauser, N. (2012). *TPC-H applied to MongoDB: How a NoSQL database performs* . Retreived 21 March,2014 from www.ifi.uzh.ch: http://www.ifi.uzh.ch/dbtg/teaching/thesesarch/VertiefungRutishauser.pdf

Truică, C. O., Boicea, A., & Trifan, I. (2013). CRUD Operations in MongoDB. Paper presented at the International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013). (pp. 347-250).

White, T. (2009). Hadoop: The Definitive Guide. O'Reilly Media, Inc.

Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. In Roedunet International Conference (RoEduNet), 2011 10th (pp. 1-5). IEEE.

US Department of Health and Human Services. (2013). The Health Insurance Portability and Accountability Act of 1996: health information privacy. US Department of Health and Human Services website. Retrieved 30 July, 2013.from:http://www.hhs.gov/ocr/privacy/.

Valley Programming. (2014). Big data datasets (large dataset examples) Boulder, Colorado. Retrieved 21 March, 2014, from www.valleyprogramming.com: http://www.valleyprogramming.com/blog/big-data-datasets-large-examples-boulder-colorado-hadoop-mongodb

Van der Veen, J. S., Van der Waaij, B., & Meijer, R. J. (2012). Sensor data storage performance: Sql or nosql, physical or virtual. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on (pp. 431-438). IEEE.

Venable, J. & Kuechler B, (2006), The Role of Theory and Theorising in Design Science Research, First International Conference on Design Science Research in Information Systems and Technology, Claremont, California, pp. 1-18.

Wisseman, S., Wilson, B., & Wichers, D. (1996). Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria. Diane Publishing Co.

Zagarese, Q., Canfora, G., Zimeo, E., & Baude, F. (2012). Enabling advanced loading strategies for data intensive web services. In *Web Services (ICWS), 2012 IEEE 19th International Conference on* (pp. 480-487). IEEE.