CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Today, the central part of every organization is data (Goyal *et al*., 2016). Owing to numerous database choices obtainable, each organization uses varied databases, each database is specific to the problem they are trying to resolve. With the innovations in the Information Technology (IT) industry, the use of Big Data, cloud computing, Web Applications and the Internet of things, the nature of demands for these databases are constantly changing. One of such demands is the need not only to manage data which is high in velocity, volume, veracity and variety but also to grant speed, scalability, reliability, continuous availability, and cost reduction and location independence (distribution) at the same time. The existing popular relational databases and some popular hybrid databases fell short of delivering these demands and so, most organizations are migrating from relational to semi and unstructured databases (Goyal *et al*., 2016). According to Lawrence (2014) "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) further noted that NoSQL systems have been propositioned to deal with applications and problem field inadequately served by relational database management systems. These domains are mainly Big data domains involving web data such as supporting millions of interactive users or performing analytics on terabytes of data such as weblogs and click streams. The data domiciled in this field of analytics is semi-structured, variable and massive.

The NoSQL systems which are open source, due to their scalable nature and design model are built to handle larger data volumes at better performance than relational systems. Another advantage of the NoSQL system is that it offers better support for programmers. Systems like MongoDB permits a Javascript Object Notation (JSON) object to be stored which can be readily converted into Javascript objects in code (BSON; binary encoded JSON). This singular advantage makes the knowledge of SQL (Structured Query Language) unnecessary for many programmers as a result of the simplicity and flexibility offered by NoSQL management systems (Lawrence, 2014).

Increasing output of data (structured, semi-structured and unstructured) have caused an upheaval in the database community. This disruption in data management stems from lack of uniformity both in syntaxes and the query language which are used to access the various database systems. Researches both in the academia and the industry have been underway to find a lasting solution for the integration/interoperability between SQL and NoSQL systems. This is as a result of the overhead which is accrued from running single APIs (Application Programming Interface) on different NoSQL systems. Also there is need for skill in the area of handling this single APIs and has called for a search into the possibility of handling different systems efficiently. Data storage has become of increasing need which has resulted in major industries, institutions and establishments seeking out their own solutions for handling big data efficiently. The available solutions so far have only addressed this issue partly and left serious gaps. Some of the gaps identified are security issues as a result of third party involvement in cloud computing, near inconsistency of the database, heterogeneous data structure and also different APIs as a result of varying query model used in the NoSQL systems which lack uniformity. These gaps, when filled, will be a major breakthrough in the field of NoSQL Database Management System.

This need for big data management has called for systems which are persistent and can handle data concurrency. According to Lombardo *et al.,* (2012), the scalability of NoSQL systems in the cloud computing field have made them an interesting dimension in IT which holds prospects for improvement as regards query optimization and execution. Though there are draw backs attending the cloud/NoSQL technology, the systems are of optimum performance individually but with proper implementation of algorithms, could be improved. This could be in form of a merge of similar systems to ascertain functionality. Furthermore, the NoSQL systems are heterogeneous in their functionality which is as a result of different query languages, different consistency models and also the difference in the CAP theorem adopted by the various systems used (Dharmasiri and Goonetillake, 2013). These various features of the systems leave a lack of proper cohesion between the systems involved which is mainly due to lack of a standard query language.

According to Gadkari *et al.,* (2014), the increasing amount of data and database accesses has rendered the traditional database systems inefficient for the purpose of big data management due to space requirements. This need for systems that can handle large volumes of data which are both non-structured/semi-structured has driven researchers and the industry into a viable option which is the NoSQL Database Management System. NoSQL in its lay term stands for 'Not Only SQL'. Ramanathan *et al., (*2011), described the recent migration for production purposes by some leading brands like Amazon and Google from relational to NoSQL databases to be as a result of its ability to handle unstructured data such as word-processing files, e-mail, multimedia, and social media efficiently. Various NoSQL systems have been developed to handle data of different formats and the four most widely known are; Key-value pair, Document-store, Column-share and Graph databases. These NoSQL systems handle data of different structures but though they have been able to meet different database needs, there are still issues attributable to the use of these various systems.

According to Cure *et al*., (2012), "NoSQL covers a wide range of technologies and data architectures for managing web-scale data and having the following common features: persistent data, non-relational data, avoid join operations, distribution, massive horizontal scaling, no fixed and flexible schemata, replication support, individual usually procedural query systems rather than using a standard declarative query language, consistent within a node of the cluster and eventually consistent across the cluster and simple transactions".

The SQL systems are highly structured in their functionality using rows and columns which are arranged in well-defined tables. Join operations are noted as one of the main features of the RDBMS (Relational Database Management System) which outputs data that have functional dependency on the key and which is well normalized to eliminate data redundancy and guarantee integrity of data (Gadkari *et al*., 2014). As stated by Hecht and Jablonski (2011), the normalized data model and full ACID support of the relational/SQL DBMS are not suitable for dealing with the web 2.0 technology. This is because web 2.0 technology requires or deals with very high volumes of data which come in peta- tera, yota, zettabytes and demands massive/concurrent read-write access which should be responded to within low latency. The joins and lock of the relational database model while ensuring data integrity is not suitable for distributed systems where data are replicated and spread across multiple nodes as a result of need for high availability of data.

Though these management systems are not very closely related and are specialized to perform specific tasks, there are some similarities in their build up. The column-share NoSQL systems display data in columns which is similar to the SQL Database Management System (DBMS). According to Lombardo *et al*., (2012) the column-share NoSQL database model is the most closely related to the SQL management system as it confines data into columns in different tables and also supports the vertical partitioning storage model.

A common ground (between SQL and column-share) could be reached in terms of data and query model. An integration of these two management systems; SQL and NoSQL which have various high points and limitations could give a system which is both highly consistent and with high throughput/availability.

Big Data Analysis

Big data as defined by Qi *et al.*, (2014) is a term which encompasses various techniques which are used to capture, process, analyze, and visualize potentially large sets of data within a judicious time frame which are not available to regular IT technologies. Big data management requires combination of different techniques which when properly managed allows for effective data management. There are three basic features of Big data which are Volume (the amount of data involved), Velocity (the rate at which data is generated) and Variety (the heterogeneous nature of data involved). As shown in Figure 1.1, there are different sources of Big data which spring from different bases or context like social media, multimedia and also cloud technology. Some works have been carried out on integration of SQL and NoSQL systems. Individual NoSQL systems like graph, document-store and column-family have been made interoperable with the SQL platform.

Different works have been conducted in this field of integration of SQL and NoSQL systems. These already existing works have not performed integration of all the various NoSQL models which are; key-value pair, column-share, document-store and graph systems with the traditional relational model which is SQL (Structured Query Language). Doshi *et al.,*(2013) classified data growth into vertical, chronological and horizontal. A solution which blends SQL with NoSQL by use of data integrator which first sifts updates in RDBMS and channels it into NoSQL was used. A framework named Hibernate-OGM used to reduce the programming complexity of the NoSQL was used to aid the relational model. This framework has the problem of handling data at good speed.

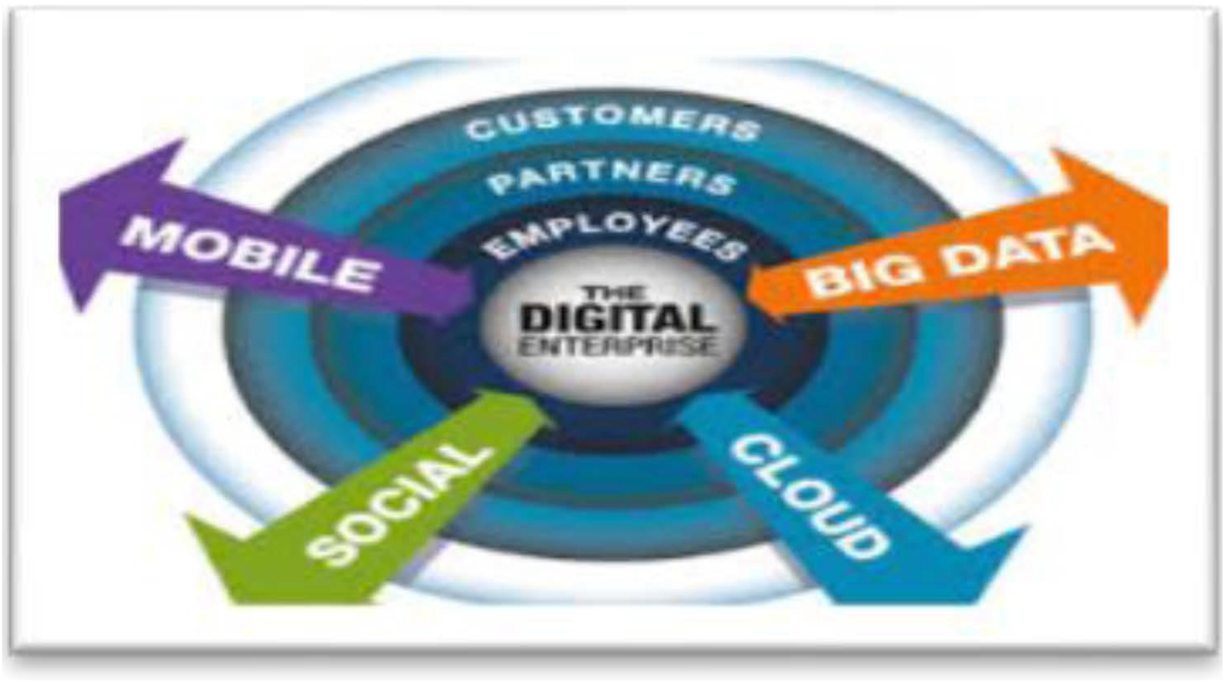


Figure 1.1 Big data source (Source: Zafar *et al.*, 2016).

Liu and Vitolo (2013) explained that the concept of ‘Graph Cube’ is proposed as a design which integrates graphs with tables. This concept serves as a prototype which is the basis of a graph data warehouse. Some DML (select, insert, update and delete) and DDL (create, drop and update) in SQL is synchronized with the graph data model to give GDML (Graph Data Manipulation Language) and GDDL (Graph Data Definition Language). This model allows for views in the graph data warehouse.

As stated by Kaur and Rani (2013) graph databases represent data in their natural format using graphical forms which shows better representation rather than tabular forms. This eliminates impedance mismatch (incompatibility of database with programming language) which is the problem mostly associated with the object-relational systems which store data in tabular form. The graph DBMS as explained by Indrawan-Santiago (2012) faces the most challenge amongst the NoSQL systems as it does not support horizontal partitioning which is a drawback to this model. As described by Gudivada *et al.*, (2014) graph database is highly scalable and has persistent in-built memory which can be used for clustered systems for both single and distributed data centers. The interoperability of both the Graph system and NoSQL system has caused an improvement in data management technology.

Lawrence (2014) worked on SQL and NoSQL integration using MongoDB and MySQL. The architecture of the system is based on the construction of a JDBC driver which accepts SQL queries through the use of SQL parser which produces a parse and relational operator tree. This process is made possible through a virtualization Execution Engine which serves as the middle-layer accepting and translating information across the two management systems.

This research seeks to serve as furtherance of the work carried out by Lawrence (2014) at the University of British Columbia Canada where he stated that "Future work involves benchmarking the performance of other supported NoSQL systems such as Cassandra. We are also working on parallelizing the virtualization engine for a cluster environment." Also, Doshi *et al* (2013) proposed further work using mixed approaches in blending SQL IMDBs with distributed, fault-tolerant NoSQL systems which demand a combination of in-memory and massively parallel computing.

1.2 Statement of the Problem

Presently, there has been so many demands by users for improvement in terms of efficiency of databases to handle large data effectively. The existing databases fell short of delivering these demands in the area of effective database management. These problems are inefficiencies in the area of speed, scalability, reliability, continuous availability, cost reduction and location independency.

Some problems have been identified by this research and they include;

i) An enhanced hybrid system capable of accommodating structured, semi-structured and unstructured datasets.

Scherzinger *et al*., (2013) explained that "what is missing in today's frameworks is a means to systematically manage the schema of stored data, while at the same time maintaining the flexibility that a schema-less data store provides". Big data management poses a serious obstacle in the IT research and industry domain in terms of the size (volume of data), variety (semi-structured/unstructured) and velocity (speed at which data changes) of the data involved. Increasing amount of data produced has rendered the traditional (SQL) systems incapable of handling these volumes of data both in speed and structure. As a result of this incapability of

SQL systems to handle different structures of data, vast volumes of data cannot be accommodated.

ii) A unified interface (integrated system) that has speed and allows seamless movement of large datasets or Big data with automatic load balancing capability.

According to Srivastava & Shekokar (2016) Big Data has lots of issues and challenges such as storage, efficient access, and data analytics and data security. Research in Big data domain which has been broadly segregated into 3 main categories; 1) infrastructure domain, 2) data processing and 3) analytics, is based predominantly on the huge volume of data which has created difficulty in providing good infrastructure with 100% uptime. Lots of cloud service providers and distributed frameworks are claiming to provide good elasticity and reliability but lots of challenges still prevail in terms of efficient load balancing and security issues.

iii) An effective system optimized for analytics and offer of good infrastructure.

Zafar *et al*., (2016) opined that the performance of NoSQL can be quantified by numerous features such as the architecture, data model, query languages, consumer API, ease of use and scalability; data in NoSQL is typically collected from different foundations and need to be handled in real time. As such, NoSQL systems do not support ACID (atomicity, consistency, isolation and durability) principle, but have a shared and fault tolerance design. As stated by Li and Gu (2019) new systems have been developed to combat this major data issue but the problem still persists. This problem is in terms of movement of data across different systems which are as a result of the different APIs employed by the different NoSQL developers. Availability of data is at the forefront of the NoSQL management system which leaves consistency of data a secondary consideration to this architecture. SQL stands for consistency and integrity of data but yet lacks the ability of horizontal partitioning/scalability as it partitions vertically. These various systems; the SQL and different NoSQL systems have their strong attributes and a proper integration has been carried out by this research and avails the research and industry with a system reliable for parsing data across the different systems involved and also provides high consistency and throughput.

1.3 Aim and Objectives of the Study

The aim of this work is to design and implement a Cross-platform Database system for SQL and NoSQL interoperability.

The specific objectives of the work are to:

1. develop an interoperable system that will increase the speed and movement of data.
2. design an integrated system capable of accommodating vast volumes of data.
3. design a novel methodology for exploiting strengths of query languages.
4. test for consistency, availability and improvement using test–case scenarios of various systems to ensure uniform communication and operation.

1.4 Significance of the Study

"In practice, there are situations in which storing data in the form of a table is inconvenient, or there are other kinds of relationship between records, or there is the necessity to quickly access the data. A NoSQL database provides a mechanism for storage and retrieval of data that is different from the typical relational model" (Stanescu *et al*., 2016).

This work is important to organizations and big corporations as it offers an interoperable system which is both fast and reliable. Data can be moved around and understood across board for the relational, NoSQL/NewSQL platforms. This new hybrid system offers full scalability, cost effectiveness, compatibility and also a consistent and available repository/storage facility.

As posited by Cure *et al*., (2012), solutions in the NoSQL ecosystem are emerging in various domains such as social, scientific and even financial applications. Nevertheless, many actors consider that in order to increase the adoption rate of the NoSQL database, NoSQL systems need to integrate some new features and these desired features correspond to the ones found in RDBMS (Relational Database Management Systems). These features were identified as the ones which are concerned with more schema, more declarative query languages and business intelligence (BI) processing. The researchers further argued that the integration of these features needs to consider the semantics of the elements of the application domain which could be a major breakthrough for both NoSQL stores and the semantic community since RDBMS is not really reactive in integrating semantics.

Lawrence (2014) put forward three primary reasons for supporting SQL querying of NoSQL systems 1) SQL is a declarative language that allows descriptive queries while hiding implementation and query execution details. 2) SQL is a standardized language allowing portability between systems and leveraging a massive existing knowledge base of database developers. 3) Supporting SQL allows a NoSQL system to seamlessly interact with other enterprise systems that use SQL and JDBC/ODBC without requiring changes. A cross-platform system which allows for interoperability between different NoSQL systems and SQL system has not been fully implemented which is the purpose of research. Though there have been various works produced by different researchers linking different SQL systems to their NoSQL counterparts, a concise interoperable system between these two school of thoughts is yet to be realized. This is the basis of this project; to design and implement a functional system which allows parsing of data across board for both the SQL and NoSQL database management systems. According to Lawrence (2014) the key advantage of supporting SQL is to allow for system portability.

This research came up with novel methodology which is a hybrid methodology for database management called 'Holistic Iterative and Incremental Approach for cross-platform modeling. This is an improvement on the already existing Iterative and Incremental model. This idea is as a result of the need to iteratively evaluate the data for both consistency and availability within a distributed data network. In the words of Kepner *et al*., (2016), "it is now recognized that special-purpose databases (such as the system proposed by this study) can be 100 times faster for a particular application than a general-purpose database".

1.5 Scope of the Study

This research covered a broad aspect of SQL and NoSQL database systems. The work looked at the various conventions/drivers which the different systems run on. SQL runs on relational algebra (using intersections and unions to perform joins) which is how the relations are put together. Polyglot persistence has been known to form a major part of how NoSQL systems run which is the combination of various techniques and technologies in managing and outputting data/information. The research was used to find a common ground between these two unique systems and how they efficiently parse/exchange and accommodate data within a unified interface. The study integrated an enhanced, functional and interoperable cross-platform database with both SQL and NoSQL peculiarities which is unambiguous.

1.6 Limitations of the Study

There are various limitations attending this study; at the foremost are time and tools to be used. The work is time-constrained as the research is required to be started and completed within a given time frame.

The researcher does not have direct access to some of the tools which will be used for the work and will depend on the limited access edition of the open source. Various techniques which will be employed for example the SQL server and client-side RoboMongo for MongoDB cannot be accessed through the University and will be sourced through external means. There are also limited existing research works on this area.

1.7 Definition of Terms

According to Darwen (2010), a Database is an organized, machine-readable collection of symbols, to be interpreted as a true account of some enterprise. A database is machine-updatable too, and so must also be a collection of variables. A database is typically available to a community of users, with possibly varying requirements.

Relational Database: - a database whose symbols are organized into a collection of relations (Darwen, 2010).

RDBMS: - Relational Database Management Systems are based on the relational model defined by a schema. This model uses two concepts: table and relationship. A relational table represents a well defined collection of rows and columns and the relationship is established between the rows of the tables. Relational data can be queried and manipulated using SQL query language and the RDBMS is mostly used for financial records, manufacturing information, staff and salary data and so on (Stanescu *et al.*, 2016).

Relation: - a formal term in mathematics –in particular, in the logical foundation of mathematics which appeals to the notion of relationships between things or any number of things (Darwen, 2010). “A mathematical definition of database tables sufficient for their representation without constraining their implementation” (Kepner *et al.*, 2016).

Relational Algebra: - an algebra with the primary purpose of providing a collection of operations on relations of all degrees (not necessarily binary) suitable for selecting data from a relational database. The relations to be operated upon are assumed to be normalized; that is, the domains on which they are defined are simple (Codd, 1972).

Relational algebra: - a set of mathematical operators that operate on relations and yield relations as results (Darwen, 2010).

Relational Table: - According to Stanescu *et al.*, (2016) the relational table represents a well defined collection of rows and columns which has relationship established between the rows of the tables. The data in the relational table can be queried and manipulated using SQL (Structured Query Language).

NoSQL: - systems developed to support application not well served by relational systems, often involving Big Data processing (Lawrence, 2014). "Typical applications of NoSQL databases involve many important areas, such as the Internet, mobile computation, telecommunications, Bioinformatics, education, and energy" (Li and Gu, 2019). Cure *et al.*, (2012) explained that NoSQL covers a wide range of technologies and data architecture for managing web-scale data and having the following common features; persistent data, non-relational data, avoid join operations, distribution, massive horizontal scaling, no fixed and flexible schemata, replication support, individual usually procedural query systems rather than using a standard declarative query language, consistent within a node of the cluster and eventually consistent across the cluster and simple transactions. Zafar *et al.*, (2016) also defined NoSQL as systems that do not support ACID (atomicity, consistency, isolation and durability) principle, but have a shared and fault tolerant design.

NewSQL: - an inclusive term to refer to a clustered, non-RDBMS system where the distinction between NoSQL and NewSQL is not important (Doshi *et al.*, 2013).

Polyglot Persistence: - is a multi-database approach of introducing different database approaches in an application. Polyglot persistence allows one database to process one part of application and use same data which is used by different database for processing another part of same application (Scott Leberknight, 2008). Srivastava and Shekokar (2016) defined polyglot persistence as where different modules can have their own different data processing mechanism, can solve the problem of scalability and processing unstructured data as well as maintaining legacy processing.

Big Data: - Zafar *et al*., (2016) defined Big data as data which have unlimited quantity of datasets and is very complex to collect and store. Srivastava and Shekokar (2016) also defined Big data as numerous sources which are creating large volumes of data such as E Commerce, search engines, IoT (Internet of Things), mobile phones, satellites etc. Big Data is not only large volume; it also includes Velocity; data changing at a very high speed, and Variety; inclusion of structured, semi-structured and unstructured data.

IoT (Internet of Things):- network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data (Srivastava and Shekokar, 2016).

DBMS (Database Management System):- a piece of software for managing databases and providing access to them. A DBMS responds to commands given by application programs, custom-written or general purpose, executing on behalf of users. Commands are written in the database language of the DBMS, for example; SQL (Darwen, 2010).

ACID: - Atomicity, Consistency, Isolation and Durability properties is used by the RDBMS (Relational Database Management System) for its operations on data. The ACID ensures a reliable performance whereby transactions are correctly changed in the database. When a transaction modifies a value in the database, other database consumers are able to see the same value that was updated (Zafar *et al*., 2016).

Normalization: - is a logical approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like insertion, update and deletion anomalies. It is a multi-step procedure that places data into tabular form, thereby eliminating replica data from the relational tables. Normalization is employed for two main purposes; 1) to eliminate redundant (useless) data. 2) To ensure accurate data dependencies (Kanade et al., 2014).

BASE: - Basic Availability, Soft State, and Eventual consistency properties are used by the NoSQL database for the operations. Basic Availability means apparent availability of data. When a single node fails, the data is partially inaccessible but other data layer remains functional. Soft

State event implies that the data can be changed over time despite the lack of input because such ability may ensure consistency (Zafar *et al*., 2016).

PACELC:- Partition, Availability and Consistency, Else, Latency and Consistency theorem inculcates all the characteristics of the NoSQL management system, due to the replication of data in NoSQL, a failure goes unnoticed as replica servers generate the exact information on the failed node. The NoSQL system considers when there is partition, availability and consistency of the database and when the system performs normally without partition, it considers latency and consistency (Indrawan-Santiago, 2012).

CAP theorem: - stands for Consistency, Availability and Partition tolerance. Professor Eric Brewer in 2000 proposed CAP theorem. According to Benefico *et al*., (2012), shared data systems have properties such as, consistency, availability and partition (CAP). This theorem states that distributed databases cannot have both consistency and availability. The CAP model grants that in a joint-data scheme, only two out of the three features can be satisfied at a particular point in time within the database. There are three possible configurations which are; consistency and partition tolerance, availability and partition tolerance and the last consistency and availability which is very difficult to combine (Bonnet *et al*., 2011).

Query: - an expression that, when evaluated, yields some result derived from the database. Queries make databases useful. A query is not of itself a command. The DBMS might support some kind of command to evaluate a given query and make the result available for access, also using DBMS commands, by the application program. The application program might execute such commands in order to display result (usually in tabular form) in a window (Darwen, 2010).

Virtualization Technology: - is an efficient method of scheduling hardware, software, data, and network, storage of the application system from each other, which breaks the physical device obstacles in the data centers, servers, storage, networks, data and applications and implements the dynamic architecture. It also provides the centralized management and dynamic use of physical resources and virtual resources, and improves the elasticity and flexibility. It can also promote the service and manage the risk (Jiang *et al.,* 2013).

Sharding: - the introduction of new node (s) to the current system which will not affect the system performance or shutdown the system but increase the capability of the system whereby data is distributed over the node and is arranged in a non-overlapped form. These nodes can be operated on different systems and as such, each node can independently eliminate source dispute (Zafar *et al*., (2016).

JSON: - JavaScript Object Notation (JSON) is a text based open source application to transfer information between server and client. JSON is derived from JAVASCRIPT. On the other hand, a binary-coded serialization (BSON) is a flavor of JSON. However, BSON is more flexible than Protocol buffer (Google's language neutral, platform-neutral, extensible mechanism for serializing structured data; useful in developing programs to communicate with each other over a wire or for storing data) but commonly employed as effective space management method (Zafar *et al*., 2016).

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of SQL and NoSQL Systems

“NoSQL and relational systems will likely co-exist for some time, and it is valuable to query them simultaneously” (Lawrence, 2014). According to Cure et al., (2012), “NoSQL stores are emerging as an efficient alternative to relational database management systems in the context of big data”. As opined by Zhao *et al*., (2014), conventionally, relational database is one of the most accepted ways to manage data in practical scene; however, faced with a substantial increase in the amount of data, relational database is suffering severe stress. Relational databases are still very much efficient and in use by companies with basic management needs that can be handled by the relational model but when big data are factored in, they are considered inadequate both in size, speed, structure and throughput. This is in part due to the schemas which confound data into columns and rows, referential integrity and constraints (primary and foreign key).

Tudorica and Bucur (2011), defined NoSQL(Not only Structured Query Language) as a distributed database system which does not require SQL (structured Query Language), does not have fixed table schemas, with no joins and can be horizontally scaled. This might be open-source. NoSQL database has been developed to cover the shortage of relational database (Hetch and Jablonski, 2011 & Han *et al*., 2011). NoSQL database offers a method for storage and retrieval of data that is modeled as unstructured other than the tabular relation used in relational databases (Zhao *et al*., 2014).

As further explained by Zhao *et al*., (2014), NoSQL database is suitable for read-heavy OLAP (Online Analytical Processing) systems where data inconsistency is tolerable such as big data scenarios and web applications; but there is a problem that the data to analyze is derived from relational database, so there is great demand for efficient and reliable solution to migrate data from relational database to NoSQL database. Many companies who have previously adopted SQL (Structured Query Language) as their primary database management system turn to NoSQL database and use it for storing and managing data, and relational database of existing applications must be transformed to NoSQL database. Data in relational database model is organized by tables and relations between tables are represented by foreign keys. NoSQL database unlike the SQL system organizes data by number of dataset and they are independent from each other. As opined by Stanescu *et al*., (2016) an additional problem that NoSQL solves is the issue of Impedance mismatch (the mismatch between relational databases and object-oriented programming). It is known that SQL queries are not suitable for object oriented data structures that are used in most applications now. Figure 2.1shows the different data models of the NoSQL systems and example of databases developed for the the purpose of handling data to conform to the model.

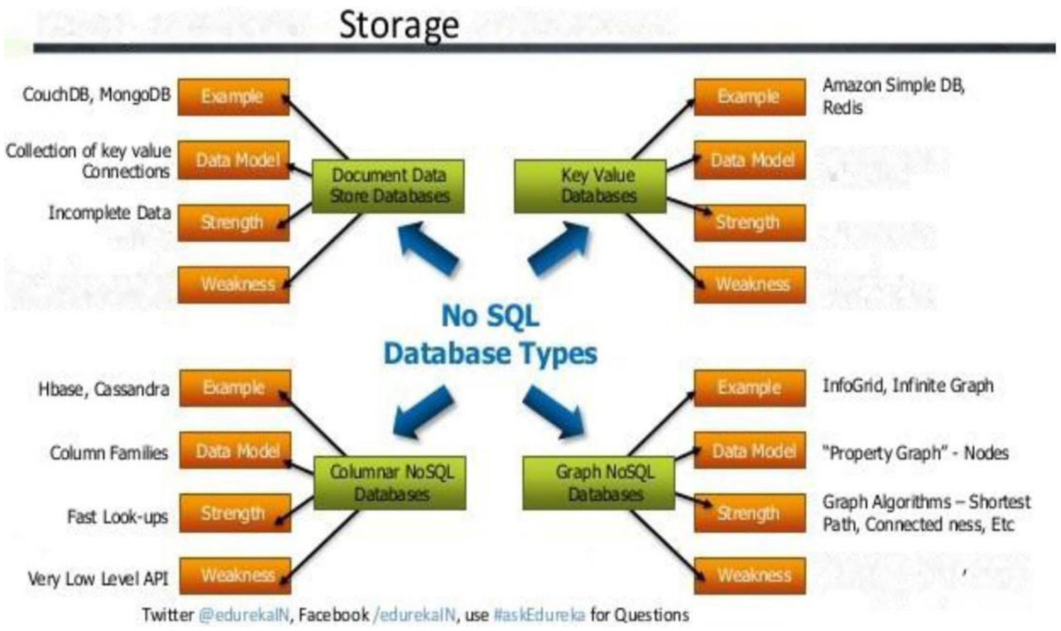


Figure 2.1 NoSQL Database Types (Source: Zafar *et al*., 2016).

According to Ejiofor and Okeke (2016), NoSQL systems were introduced as feasible alternatives to the RDBMS in mitigating the difficulty experienced as a result of large data sets. These data sets come in structured, semi-structured and unstructured forms thereby rendering the well-defined data type formation of the SQL systems impracticable for the storage of large sets of data. The necessity for fast storage and high availability of data with ensuing quicker recovery of the stored information also supported the concept of NoSQL and big data systems development.

Table 2.1 shows a listing of the different NoSQL systems rnaging from key-value pairs to document-store, with the column-share and graph-oriented data models. The table 2.1 depicts the various configurations and classifications of already functional NoSQL systems and their use-case similarities.

Table 2.1Classification by data model of different NoSQL databases (Source: Zafar *et al*., 2016).

Data Model	Example Databases
Key-Value	Berkeley DB LevelDB Memcached Project Voldermort Redis Riak
Document Store	CouchDB MongoDB OrientDB RavenDB Terrastore
Column Family	Amazon SimpleDB Cassandra HBase Hypertable
Graph	FlockDB HyperGraphDB Infinite Graph Neo4J OrientDB

The traditional database management systems also known as the relational database management systems (RDBMS) which make use of SQL (Structured Query Language) for its IO (Input/Output) operations have been around since the 1970s. It was developed by Jeff Codd and has been known to offer highly consistent data. This format of data management confines data into rows and columns (tables) which are known as schemas. This data is highly structured and dependent on the primary key which is used to extract the data. Various vendors have their own type of query language which are peculiar to them and differ slightly in executed features but they all have one thing in common: the syntaxes are similar and are made to coordinate structured data. Also, the relational database systems are relatively interchangeable due to their mutual recognition of standards (Lawrence, 2014). Gyorodi *et al*., (2015), explained that relational models are extensively used in most of the applications and they have good performance when they handle a restricted volume of data. In handling huge amount of data like internet, multimedia and social media, the use of traditional relational databases is ineffective. A comparison of the different hierarchies of MySQL and Mongo DB database and their equivalence when juxstaposed with each other is highlighted in Table 2.2.

Table 2.2 Tabular representation of MySQL and MongoDB systems. (Source Gyorodi *et al*., 2015).

MySQL	**MongoDB**
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded documents and linking
Primary key	Primary key
Group by	Aggregation

According to Stanescu *et al*., (2016) the main concepts in MongoDB are collection and document. Database is a physical container for collections. Each database obtains its own set of files on the file system. A single MongoDB server typically manages multiple databases just like the MySQL server would do. The collection is a set of MongoDB documents which is the equivalent of a relational table in the MySQL RDBMS. The collection exists only within a single database. A MongoDB document is a set of key-value pairs which could be compared to rows in the relational model. The documents do not necessarily have to be schema compliant.

Due to recent surge in data output which was brought about by the advent of web 2.0 technology, it has become pertinent on data managers to inculcate into their management, all kinds of data which ranges from multimedia to voice and these data are not all structured. Some of the recent data are semi-structured in nature while others are unstructured. The problem does not end at this; these recent data are coming in large quantities and speed and the users of the data in question demand real time answers to data processing; their availability has been placed in the front-burner of data management. This is the reason why researchers both in the industry have gone in search for means to handle this huge task and they have come up with NoSQL (Not Only SQL). The NoSQL database management system as posited by Lawrence (2014), "are growing in popularity for Big Data applications in web analytics and supporting large web sites due to their high availability and scalability". As posited by Ramanathan *et al., (*2011), the recent migration for production purposes by principal brands like Amazon and Google from relational model to NoSQL databases is as a result of its capability to handle unstructured data such as multimedia, word-processing files, e-mail, and social media proficiently. Increasing necessity of companies and database users for other options which could aid data storage needs compelled developers to conceive the idea of NoSQL databases. Developers of NoSQL databases use diverse approaches which is geared towards the same end purpose which is that they are not relational (structured). NoSQL databases leverage high performance both in terms of speed, size and high availability at the expense of losing the ACID (Atomicity, Consistency, Isolation and Durability) features which are the key characteristics of the RDBMS. The NoSQL systems are modeled to chiefly conform to features of key-value, column-orientation, document-store and graph databases. Different NoSQL databases make use of these properties according to the need of the developer and in line with the purpose the system was built to serve (Tudorica and Bucur, 2011).

According to Lombardo *et al., (*2012) NoSQL DB is principally organized in uncorrelated tables. This allows for a multi-layered, fundamentally relational data model which is flattened and subdivided into various NoSQL tables. Two different partitioning strategies are probable; vertical partitioning where each column of a high level relational table is stored into a separate NoSQL table and horizontal partitioning (sharding) where different sets of values are stored in different NoSQL tables. The NoSQL database allows both forms of partitioning.

2.2 Comparison of SQL and NoSQL Databases

According to Zafar *et al*., (2016), Netflix converted its data management system from Oracle (RDBMS) to Cassandra (NoSQL system) as a result of the Cassandra's ability to provide reliability, readiness and fault tolerance in handling big data. After the migration to Cassandra by Netflix, the company achieves 10,000 writes per second per node and the average latency rate is less than 0.015. The total cost of Cassandra set for running on the Amazon EC2 is $60 per hour for a cluster of 48 nodes which shows that NoSQL system are built to support read and write operations effectively.

As stated by Cure *et al*., (2012), NoSQL covers a wide range of technologies and data architectures for managing web-scale data and having the following common features; persistent data, non-relational data, avoid join operations, distribution, massive horizontal scaling, no fixed and flexible schemata, replication support, individual usually procedural query systems rather than using standard declarative query language, consistent within a node of the cluster and eventually consistent across the cluster and simple transactions.

SQL systems offer superior consistency/isolation as opposed to the NoSQL system which trades consistency for availability. This is due to real-time needs of day-to-day users who require response to answer for transactions being performed (Indrawan-Santiago, 2012). As stated by Lawrence (2014), the relational systems have been the dominant form of data storage and manipulation for the last 30 years; this is because the RDBMS is both theoretically grounded and efficient to implement. Relational databases are good for their provision of data integrity and offer of big feature set; which are not attributes of the NoSQL databases. These features of the traditional database systems have overhead cost and complexity which are attached to them (Pettersen *et al.,* 2014). Data integrity and the offer of big feature set, key attributes of relational databases, are a handicap of NoSQL databases. As argued by Hecht and Jablonski (2011), the provision of the large feature set by the relational model can amount to needless overhead when the query is being performed for simple tasks such as logging. The traditional databases also known as relational databases have some qualities attributed to them like atomicity, consistency, isolation and durability. Bonnet *et al.*, (2011), explained that the ACID model of the RDBMS ensures the database is very consistent and can perform optimally with the transactions fully committing or none at all. Hecht and Jablonski (2011) argued that the data model in the relational databases which are normalized and its full support for the ACID properties can affect the performance of the database negatively when used for recent web activities like Facebook as joins and locks cannot perform well in a distributed environment. Joins also consume considerable resources to implement and are costly to write, debug and maintain (Gudivada *et al.*, 2014).

According to Lawrence (2014), NoSQL systems are frequently open-source and are aimed to handle larger data capacities at improved performance than the relational systems; another key benefit of the NoSQL systems like MongoDB is that they leverage better support for programmers. MongoDB permits a JSON object to be stored which can be simply transformed into JavaScript objects in code. This ease and flexibility makes using MongoDB easier for many programmers thereby eliminating the need to understand SQL.

According to Han *et al.,* (2011), other advantages of the NoSQL database is its ability to read and write data quickly and support for mass storage, ease of expansion and low cost. In spite of these benefits, there are some inadequacies linked to NoSQL databases which are;

1. Lack of support for SQL (Structured Query Language): the SQL is the industry standard for querying and updating a database and the NoSQL systems do not make use of it.
2. Lack of transactions: transactions are the basic unit of work of a database. In the NoSQL model, there are no transactions which could be used to assess the functioning of a database.

Another setback of the traditional database is its inability to perform large amounts of read and write concurrently (Tudorica and Bucur, 2011). Being a very consistency conscious database, the relational database performs its read/write operations slowly to ensure there is no loss of information (Xiang *et al*., 2010).

The employee_records table as shown in Table 2.3 is fully normalised which means it is in third normal form and there is no existence of transitive dependencies within this relation. Data is held in rows in this table and there are different constraints applied to ensure integrity of this table. For example there is a primary key constraint (number (10)) on the employee_id row which means number cannot exceed 10. Also because the employee_id is the primary key, the values of all the other attributes: first_name, last_name, position and salary are functionally dependent on it.

Table 2.3 Example of a structured database table for Employees records in Relational database (Source: Ejiofor and Okeke, 2016).

Employee_id	First_Name	Last_Name	Position	Salary in USD
1	David	Okaro	Systems Analyst	25,000
2	Jerry	Orakwe	Manager	30,000
3	Musa	Ahmed	Accountant	23,000

Example of Table creation and query in ORACLE SQL

Drop table employee_records;

Create table employee_records

(employee_id number (10) not null primary key,

```
First_name varchar2(15) not null,

Last_name varchar2(15) not null,

Position varchar2(25) not null,

Salary varchar2(10) not null);

Commit;

Insert into employee_records values (1, ‘David’, ‘Okaro’, ‘Systems Analyst’, ’25,000’);

SELECT employee_id, first_name, last_name, Position, Salary

From Employee_records

Where Office_no = ‘RM501’;
```

Data for employee records as depicted in table 2.4 is unstructured. There is no defined schema holding the values together. For example in key:1 there is ID: zx and First Name: Yinka while key:2 holds values for Email, location and Age. Values can be generated using the keys. The contrast between the structured and unstructured model is that the different entities like ID, First Name, location and Age would be created for both key 1 and 2 in the structured table to ensure integrity of the keys.

Table 2.4 Example of unstructured data for Employee records (Source: Ejiofor and Okeke, 2016).

Key: 1	ID: zx	First Name: Yinka

Key: 2	Email: yinka_afolabi@sec.org	Location: Lagos	Age: 35

Key:3	Facebook ID: Adisa James	Password: infosectm	Name: Afam

Syntaxes in MongoDB NoSQL

```
db.employee_records.insert( { 'employee_id': 1, 'first_name': Yinka', 'last_name': 'Williams',
'Job_title': 'Software Engineer', 'Office_no': 'RM501' } );//used to insert and save data//
```

```
db.employee_records.find(); //This is the select function in MongoDB database and can be used
                              for data retrieval//
```

```
db.users.find()sort( {“Job_title”: Software Engineer} ); //This is an equivalent of the 'where'
clause in SQL//
```

```
db.employee_records.drop(); //This is used to drop a table in MongoDB//
```

2.3 Polyglot Persistence

Scott Leberknight (2008) defined polyglot persistence as “a multi-database approach of introducing different database approaches in an application”. The concept of polyglot persistence is to let one database to process one part of application and use same data which is used by different database for handling another part of same application. Polyglot Persistence as posited by Srivastava and Shekokar (2016) is where different modules can have their own different data processing mechanism (using diverse data storage technologies to handle varying data storage needs). Polyglot persistence can also be used to solve the problem of scalability and processing unstructured data as well as sustaining legacy processing.

The whole idea is basically that of interoperability of the different database management systems which is aimed towards improved performance and optimal throughput. Polyglot Persistence can be applied across an enterprise or within a single application. The technology encapsulates data access into services thereby reducing the impact of data storage choices on other parts of a system. Big data analytics which handles real life applications has proven single database management structure insufficient whether it is a cloud based structure or any other distributed framework (Srivastava & Shekokar, 2016).

According to Ejiofor and Okeke (2016), NoSQL allows enhanced and upgraded performance which is very essential for applications with large volumes of data. Big companies like Amazon and Google have industrialized their own NoSQL database versions to embrace their increasing data and infrastructure needs. Amazon developed the Dynamo distributed NoSQL system which has SimpleDB as the web interface whereas Google has the Big Table NoSQL database. Cassandra (column-oriented NoSQL system) was developed by Facebook to support its website while Apache generated the CouchDB; a very scalable and open-source system which could be accessed from any browser. These numerous databases were fashioned to solve various needs by the developers which are as a result of the development of the web2.0 technology. The relational databases cannot be used to accomplish multi-table join queries where enormous data is involved (Wei-ping *et al.,* 2011).

As posited by Srivastava and Shekokar (2016), E-commerce is one of the fastest growing business models in this era of Big data and this is aided by advancements in infrastructure technologies and data processing techniques which inculcates the idea of polyglot persistence. The relational database model still maintains wide usability as a result of ease of implementation and also due to its evolved state but the need for scalability and data processing for unstructured data which is not handled properly by the RDBMS has proven the relational model insufficient for present day business transactions models. “Scalability is the ability of a system to handle growing amounts of data. Vertical scalability (a.k.a) scaling up is easier to achieve as compared to horizontal scalability (a.k.a) scaling-out. As the name suggests, scaling-up means adding up resources to a single node and scaling-out means adding more nodes to a system. Horizontal scaling provides more flexibility as commodity servers or cloud instances can be utilized. Traditional databases relies on vertical scaling whereas recently evolved non-relational databases use horizontal scaling for achieving scalability” (Kaur and Rani, 2013).

On the other hand, applying one NoSQL for all the modules of E-Commerce is impractical as there are different requirements for the various modules; some modules require consistency while others require availability (Srivistava & Shekokar, (2016). This is because different NoSQL systems employ different partitioning techniques, two different partitioning strategies are probable; vertical partitioning where each column of a high level relational table is stored into a separate NoSQL table and horizontal partitioning where different sets of values are stored in different NoSQL tables.

According to Neal Ford (2006), application of the theory of polyglot persistence where different modules can imbibe their own different data processing apparatus can alleviate the difficulty posed as a result of scalability and processing unstructured data as well as maintaining source processing. Zhao *et al*., (2014) noted that relational databases have been faced with substantial increase in the amount of data and are undergoing severe stress as a result of the explosive growth in the size of dataset and this has attracted increasing attention to big data processing. These storage needs pushed researchers of denormalization into looking for methods to design relational database schemas that can enhance read performance and also guarantee consistency of the database when modifying data at the same time. SQL databases have been underscored for their lack of provision for techniques to effectively store and manage different types and structures of data and also for their inability to provide scalability and reliability for these types of data which the NoSQL systems have been developed to cover for. Also, the NoSQL systems offer a mechanism for storing and retrieving data which are unstructured other than the tabular relation used in RDBMS.The NoSQL systems are notable for their suitability for OLAP (Online Analytical Processing) systems which are read-intensive where short data inconsistency can be endured which plays out in web applications and big data scenarios. Zhao *et al*., (2014) explained that these data to be analyzed always arise or are derived from relational databases and for this reason; there is great demand for efficient reliable solution to transfer data seamlessly from relational to NoSQL database.

2.3.1 Issues of Polyglot Persistence

As stated by Srivastava and Shekokar (2016), there are three major issues which attend the successful implementation and maintenance of a polyglot approach in E-Commerce and they are: Integration, Access Control and Authentication.

Integration

The various NoSQL management systems have differing commands and instructions which are employed in their processing i.e. API (Application Programming Interface). The concept of integration is paramount to the successful implementation of E-Commerce as the various systems involved needs to communicate flawlessly with each other.

Access Control

The insistent problem with NoSQL database is the nonexistence of data governance. This is as a result of demand for high availability of data which needs to travel speedily once there is a demand for it. Handling E-Commerce transactions encompasses a model where data flows through customers to merchants, banks and also courier service, certifying right access to all these entities which is a demanding task for all the NoSQL systems used.

Authentication

NoSQL systems are deficient in their security buildup as they do not have their own security mechanism. The security measures employed are simplistic in approach and sometimes use external methods such as LDAP or Kerberos which are not very sustainable security systems for access enforcement. This lack of strong authentication mechanism for polyglot persistence makes finding a fool proof authentication system a good topic for research.

2.4 Different SQL Implementations

Table 2.5 shows the different versions /implementations of Structured Query Language and their developers.

Table 2.5 Different SQL Systems Implementation

Standard	The latest official version of SQL is SQL:2008.
PostgreSQL	PostgreSQL 8.4.1 on CentOS Linux.
DB2	DB2 Express-C v. 9.1 on Fedora Linux.
MS SQLServer	MS SQL Server 2005 on Windows XP. Microsoft's SQL implementation is sometimes named *Transact-SQL*, or *TSQL*.
MySQL	MySQL Database Server 5.0.18 on Fedora Linux
Oracle	Oracle Database 11*g* Release 2 on Red Hat Enterprise Linux.
Informix	Informix Dynamic Server Workgroup Edition v. 11.50 on Red Hat Enterprise Linux

2.4.1 Advantages of SQL

There are various features which are attributable to the RDBMS which makes it unique both in structure and in performance. These various characteristics of the relational database systems have made them stand out over the years. Some of these features are;

Query Optimization

According to Srivastava and Shekokar (2016), the SQL systems have query optimization engines which help them maximize query processing. Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans. The query optimizer cannot be accessed by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.

The query optimization engine is not obtainable in the NoSQL database and this is as a result of the different query processing and different APIs (Application Processing Interface) of the NoSQL systems (Srivastava and Shekokar, 2016).

ACID Property

Atomicity: In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are.

Consistency: A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.

Isolation: A transaction in process and not yet committed must remain isolated from any other transaction.

Durability: Committed data is saved by the system such that, even in the event of failure and system restart, the data is available in its correct state.

Lawrence (2014), explained that there are some major aims for supporting SQL querying over NoSQL systems:

i. SQL model is a declarative language that tolerates descriptive queries whereas hiding implementation and query execution details.
ii. SQL is a consistent language allowing transferability between systems and leveraging an enormous prevailing knowledge base of database developers.
iii. Supporting SQL permits a NoSQL system to effortlessly co-operate with additional enterprise systems that use SQL and JDBC/ODBC without requiring changes.

As further expounded by Lawrence (2014), the crucial advantage of supporting SQL is to allow for system portability, this is in contrast to NoSQL systems which must use a custom API and query language in accessing a database. Lawrence (2014) explained that there are benefits accruing from implementation of a virtualization system that has a relational query processor that can execute portions of SQL queries that cannot be executed by a NoSQL system and they are:

iv. Allows support for operations that cannot be executed by the system such as joins operations and enhanced filtering.
v. Detaches users and programmers from writing convoluted data manipulation code to implement operators that are not supported by the system.
vi. Permits data movement and querying between NoSQL and relational systems.

2.4.2 Limitations of SQL

Static Schema

Zafar *et al*., (2016) stated that "relational database works on the basis of static schema and the schema matures with the time. If the schema has to be changed, the whole application has to go through the stress of software testing". Alternatively, in the NoSQL system, the schema develops with the passage of time. The NoSQL offers a flexible schema for the data stored in row including the NULL value complications that insistently occur in the relational model. According to Stanescu *et al*., (2016), any relational database has a certain design schema that shows the tables and the relationships between them. The schema shields the data within it and when queries are run, it throws up error when the information required is not contained within the tables. It also returns error for the wrong use of query language. As Zafar *et al*., (2016) opined, conservatively, the database in RDBMS is created based on the relational data model. The RDBMS conforms to the description of data structures, storage and retrieval and reliability. In relational databases, data is stored in tables, which consist of rows and columns. To lessen data recovery issue, data in the table is not repeated. For that purpose, RDBMS is reliant on primary keys, where each row is assigned with a distinctive row number.

Lack of support for semi-structured and unstructured data

The SQL databases do not handle data beyond the structured data. Unstructured/semi-structured data are considered ambiguous both in theory and application for the SQL management systems. This is because of the coordination and retrieval process of the database. As opined by Zafar *et al*., (2016), the relational database model includes consolidation of tables which makes the querying mechanism very complex. As a consequence, the RDBMS consumes high resources to maintain and the system is expensive.

Complex Join Operations

Zafar et al., (2016) explained that RDBMS offer convoluted join operations like the inner join, outer joins etc. which are compound in their functionality. This complex joins slows down the performance and speed of the database management system as a result of the overhead cost incurred by running such operations.

CRUD Operations

According to Zafar *et al*., (2016), the intricate tasks in relational database management system are the Create/Insert, Read, Update and Delete (CRUD) processes. Contrary to the CRUD operations, the NoSQL databases are characteristically designed to accommodate a large volume of read and write operations. Despite functioning on a substantial amount of data, the fundamental attribute of NoSQL databases is its capability to offer such services (read/write operations) efficiently while maintaining the data integrity. The processes in NoSQL have been shown to function proficiently as against the update and delete operations in RDBMS.

Scale-up processing/NULL value problem

SQL databases offer only vertical scaling of data while NoSQL offers scale-up/scale-out (horizontal scaling or sharding). This means that data in the relational model can only be stored in a stringent format according to laid down specifications which conform to the table schema (Doshi *et al*., 2013).

Zafar *et al*., (2016), explained a null value to be basically an undefined value. The use of null values is to ensure that rows in the table are not left empty which is satisfactory at first but redundant in the long run which incurs overhead for the relational database model (Stanescu *et al*., 2016).

Figure 2.2 shows a schematic description of the Relational Database Management System (RDBMS). The entities and relationship which make up the SQL management are depicted in Figure 2.2 for easy understandability.

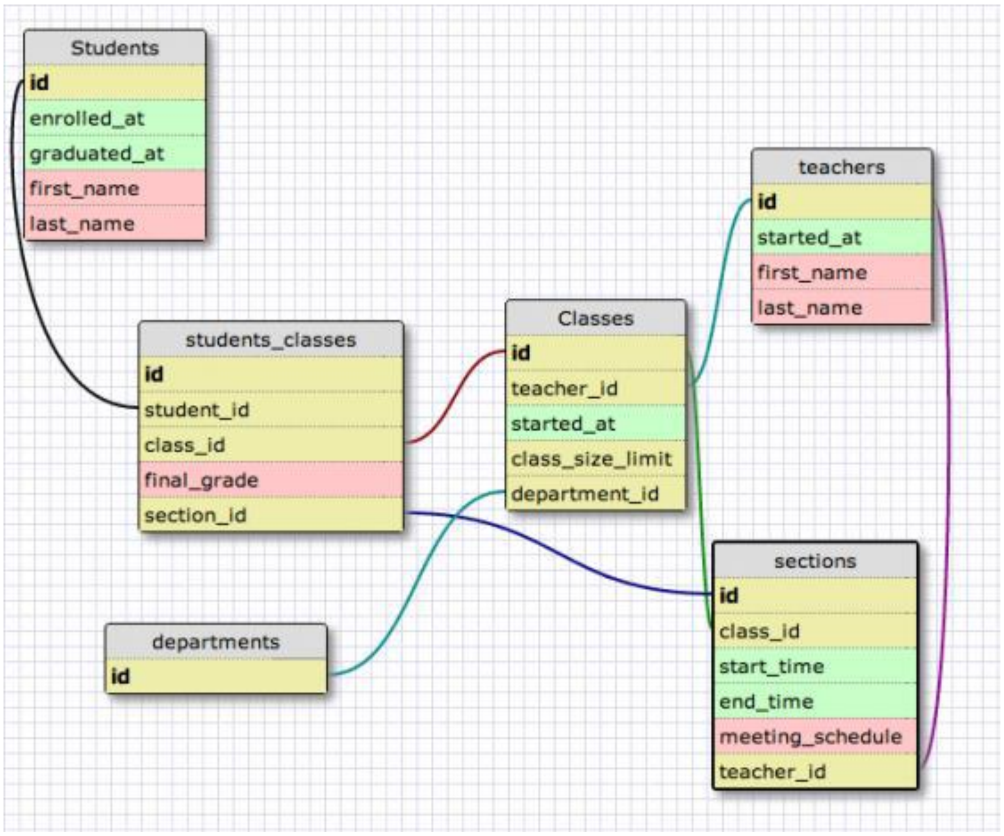


Figure 2.2 RDBMS scheme (Source: Zafar *et al*., 2016).

2.5 Types of NoSQL Databases

The NoSQL databases are generally divided across four major categories and they are the key-value store, the column family, document store and the graph databases (Jayathilake *et al*., 2012). These databases and their various attributes will be discussed in detail below.

Key-value Store

According to Srivastava and Shekokar (2016), "a key-value store is a NoSQL database that is presented by a Key (an id, a name), which returns an arbitrary large data (the value)". The concept of key-value store is based on storage of data as a pair; the key and the value (Bonnet *et al*., 2011). Key-value has no language like relational databases. As explained by Gyorodi *et al*., (2015), "key-value databases such as Riak, are conceptual distributed dictionaries and do not have predefined schema; they are schema less. The key can be synthetic or self-generated, and the value can be anything: string, JSON, BLOB and others". The key/value is made up of two columns; one column contains the key which represents the element while the other is made up of the value which is the different parts of the element and they are data stored by a primary key (Jayathilake *et al*., 2012). It can be considered as a dictionary which has a lot of words with many definitions. As posited by Srivastava and Shekokar (2016), Amazon is the best example of a key-value online shopping system as the shopping portal provides scalable, reliable and always available services. The key-value store database is more efficient when it makes use of the virtualization technology which grants it unlimited software licenses and provides lot of ease for the developers. Srivastava and Shekokar (2016) argued that although key value stores provide lots of advantages in database management, it cannot be used efficiently if 1) there are many different relationships of data, 2) multi-key transactions, 3) operations by sets and 4) queries are based on values. They further stated that the key value stores can work very well for use cases such as 1) shopping cart application, 2) user profile information and 3) session information.

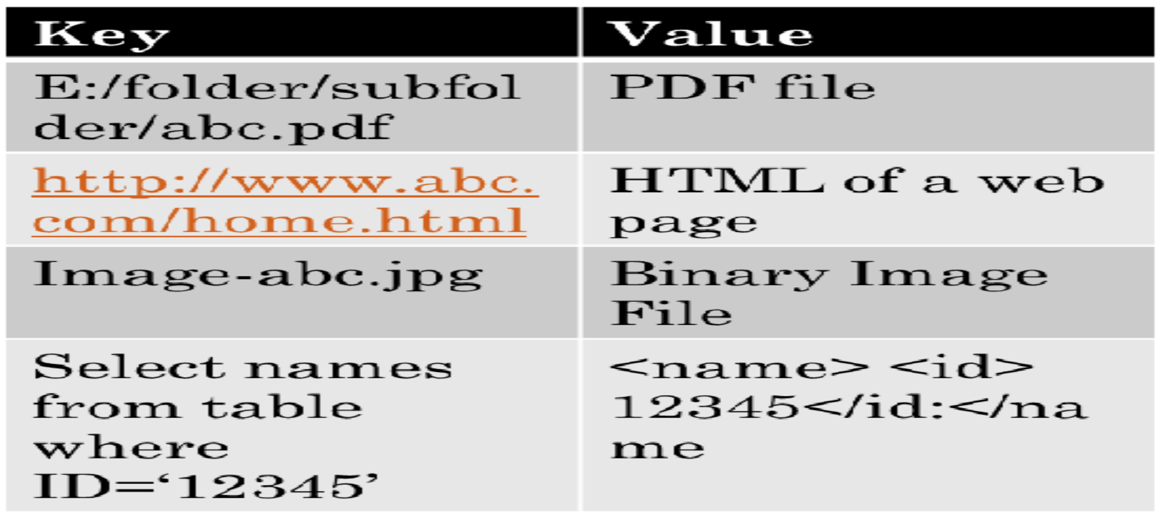
As noted by Li and Manoharan (2013), NoSQL databases are modeled for improved key-value stores. The structure of the key/value pair contains one or more attributes which makes it similar to the data dictionary model of the relational database. Access to the value is gained through the association of the key and the value. And the key is always uniquely identifiable in a collection (Indrawan-Santiago, 2012). This means that there is only one key. Han *et al.,* (2011), identified the key-value model as the correspondence or mapping of a value to a key. This structure is simplified and allows faster query operations and modifications through the primary key. Read/write operations are performed faster in the key/value pair model as it allows for concurrency. Nyati *et al*., (2013) noted that access to the key/value pair is purely through the primary key using methods such as PUT/GET/DELETE.

Example of Key-value store (Redis)

Redis is an example of the key-value store type of NoSQL database. Gudivada *et al*., (2014) defined Redis as a key-value store NoSQL system which has in-built memory, offers high availability, can carry out backup and recovery and also supports ACID (atomicity, consistency, isolation and durability) transactions due to its use of primary key. Han *et al*., (2011) identified some features of Redis as; periodic asynchronous caching to hard disk after the data has been committed to memory which is in-built in the Redis system and support for various operations like List and set which help in maintaining accuracy within the model. Han *et al*., (2011) further noted that Redis can only hold a maximum value of up to 1GB and the main shortfall is its use of physical memory which has low capacity and therefore cannot be used for storage of big data. This lack of capacity for storage of big data reduces performance due to poor scalability of the system (Ejiofor & Okeke, 2016).

Table 2.6 has all the values held in the key and the value columns. The key is used to insert, generate and delete the information in the value column.

Table 2.6 Example of a Key-Value Store (Source: Srivastava and Shekokar, 2016)



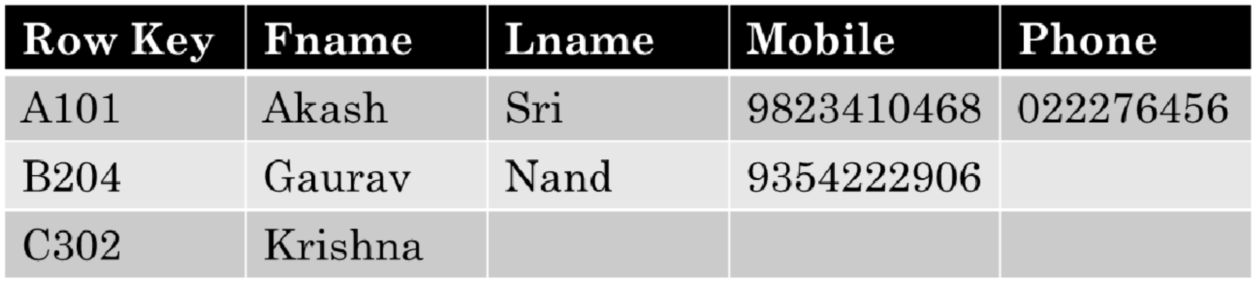
Key	Value
E:/folder/subfolder/abc.pdf	PDF file
http://www.abc.com/home.html	HTML of a web page
Image-abc.jpg	Binary Image File
Select names from table where ID='12345'	<name> <id> 12345</id:</name

Column-family

As explained by Srivastava and Shekokar (2016), the column-share NoSQL system stores data with a unique column. This type of NoSQL database adopts the vertical partitioning type of storage model where data are modeled to fit into columns of different NoSQL tables which make it similar to relational database model (Lombardo *et al*., 2012). The unique column data will have different fields connected; some of the column data may have 4 fields filled while others may have 3 fields filled. This is because the column family does not depend on join operations like in the relational model so they are not compelled to fill up all the fields because they can scale well. This technique allows data to be accessed through columns or column-groups. The difference between the column-store and the relational model is that tables are stored in rows for the relational database while column-oriented NoSQL databases store data in detached NoSQL tables. The column based databases are ideal for joint data storage mostly for the versioned data as it offers time stamping utility, also, the column NoSQL system is efficient and can sustain big scale data operations such as categorization, transformations, analyzing and algorithmic operations (Zafar *et al*., 2016). Lombardo *et al*., (2012), stated that the dissimilarity between these two models of data segregating is based on their physical infrastructural layout where data is stored in rows in the horizontally partitioned model and the column family databases where data is stored in columns. Column family stores are best suited for use cases such as event logging and expiring usage (a website which is live for a duration of time, for example a website for an event) but should not be used where schema keeps changing or processing requires Sum, Average like functions (Srivastava and Shekokar, 2016).

As shown in Table 2.7, data is stored in columns in this model. The information in these columns would have been normalized and stored differently in a relational database as all the information would be saved in rows.

Table 2.7 Example of Column Family store (Source: Srivastava and Shekokar, 2016)



Row Key	Fname	Lname	Mobile	Phone
A101	Akash	Sri	9823410468	022276456
B204	Gaurav	Nand	9354222906	
C302	Krishna			

Example of Column-family (Cassandra)

According to Cure *et al*., (2012), “Cassandra is a column family database having a data model that is dynamic and column-oriented”. Cassandra is an open source database that adopts the BASE (basically available, soft state and eventually consistent) model as it favours availability of data over its consistency and follows the key-value store arrangement while providing high scalability across a distributed system (Wang and Tang, 2012). Cassandra implements more write operations than it does read operations. This write operation is executed without checking the consistency state of the database and uses system versioning process of data to guarantee consistency when carrying out read operation. This leads to a reduced consistency level in the database (Bonnet *et al*, 2011). As explained by Cure *et al*., (2012), Cassandra allows fast lookups, and provides support for ordered range queries. Cassandra is acknowledged to be really fast for writes in a write-intensive environment but the read operations are slower than the writes. Cassandra supports secondary indexes, i.e. index on column values (Cure *et al*., 2012).

Graph-oriented Databases

Srivastava and Shekokar (2016) stated that graph store has three properties; nodes, edges as relationships and properties and can be used for solving business problem that has complex relationships between objects such as social networking and rule-based data. The graph-oriented NoSQL system can be queried to ask questions like; what is the shortest path between two nodes in a graph? What nodes have neighboring nodes that have specific properties? Given any two nodes in a graph, how similar are their neighboring nodes? What is the average connectedness of various points on a graph with each other? Castelltort and Laurent (2013) defined graph-oriented NoSQL database as the interconnection between vertices (nodes) which are linked by lines from edges. This is the joining of nodes to edges to produce an element. The nodes form the graph when they are linked with relationships. This relationship between different nodes is what produces the graph and two different nodes; the start node and end node are joined together by a relationship.

As stated by Zafar *et al.*, (2016), the graph-oriented databases are useful when persons are involved in the data relations rather than the original data; for example, a user is interested to see the correlation of data in multiple social networks, getting references or to conduct some medical analysis. As posited by Kaur and Rani (2013) graph databases represent data in their natural format using graphical forms which shows better representation rather than tabular forms. This eliminates impedance mismatch (incompatibility of database with programming language) which is the problem mostly associated with the object-relational systems which stores data in tabular form.

According to Indrawan-Santiago (2012), the graph database faces the most challenge amongst the NoSQL databases due to its lack of support for horizontal partitioning. The horizontal partitioning which is one of the major attributes of the NoSQL databases cannot be performed in graph databases. This has been identified as a drawback in the graph-oriented NoSQL systems. The graph systems as shown in figure 2.3 are composed of relationships which are either static or dynamic. These relationships show connections between objects which is known as connected data for example Google and Facebook are connected data. Gudivada *et al.*, (2014) further stated that graph systems have built-in access control systems used for authorization on subsystems for applications designed for large number of end-users for example airlines and healthcare industries.

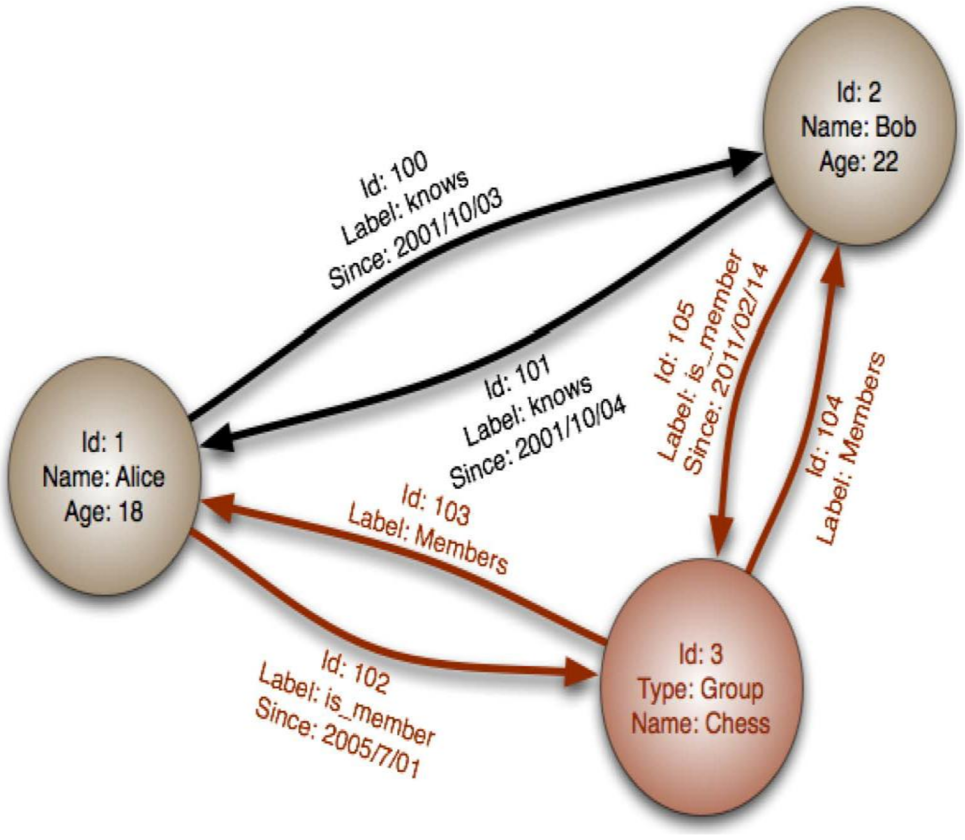


Figure 2.3 Graph-based database (Source: Zafar *et al*., 2016).

Example of Graph Database: Neo4j

Kaur and Rani (2013) described the Neo4j as NoSQL model which is written in Java and complies with ACID properties which means it is transaction compliant. The nodes and edges which connect to the node form the network structure and are equivalent to the entities and relationships of the relational database model. Neo4j as described by Gudivada *et al*., (2014) is highly scalable and has persistent in-built memory which can be used for clustered systems for both single and distributed data centers.

Document-store Databases

According to Cure *et al*., (2012), document databases emphasize on storage and access enhanced for documents as opposed to rows or records. The data model is basically the collections of documents which are comprised of key-value collections. Document-store databases are predominantly developed for large data storage. It is a schema-free type of NoSQL database which provides support for storage of unstructured and semi-structured data and these documents are stored in JSON (JavaScript Object Notation) format. The document-store NoSQL databases allow creation of convoluted data structure like arrays as an individual database entry which can be retrieved within one read procedure (Nyati *et al.*, 2013). As explained by Cure *et al*., (2012), "in a "document" the values can be nested documents or lists as well as scalar values; names are not predefined in a global schema but dynamically defined for each document at runtime".

The document-oriented model makes every document identifiable by a unique or special key which is identified as "ID". This helps in managing its compound data structures as if they are objects linked together. The document-oriented database handles high concurrency read/write operations with ease and can access big data while simultaneously providing high availability and scalability of data. Another feature of this NoSQL system is the use of nested documents in arranging data and increasing the accuracy of stored data. According to Jiang *et al*., (2013) nested document views a document to be the value of another document. This makes a connection between data and is arranged in hierarchy with record as the root of a document-tree which branches out to form a schema-free database and this relationship between the record and the document-tree forms the nested document. This arrangement in a defined structure helps improve and maintain accuracy within the database.

As noted by Okman *et al*., (2011), there are security issues attached to the document-store database system and this is as a result of the broad range of applications which it handles.

NoSQL systems have been identified by Grolinger *et al.*, (2014*)* as the database which is most suitable for Big data management because it can hold huge amounts of data in any format.

Figure 2.4 shows a hierarchical conception of the document-store NoSQL sytem.

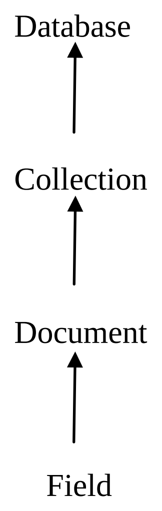


Figure 2.4 Layout of the Document-store Model (Source: Cure *et al*., 2012).

According to Srivastava and Shekokar (2016) document store databases are suitable for 1) stock market 2) finding user logged to your website (finding real time analytics and 3) content management (blogging). Srivastava & Shekokar (2016) argued that document store management systems are not suitable if application has complex transactions that support ACID (atomicity, consistency, isolation and durability) transactions or queries keep changing very frequently. In the Document-store NoSQL Management System, the database holds data in a collection which is similar to tables in the RDBMS. The collection stores a list of related documents which is the equivalent of rows in the NoSQL database and the document has limitless fields which can be added to document. The field takes semblance of the column in the SQL model (Bonnet *et al*., 2011). The difference between the relational and non-relational model is that the collection is schema-less thus eliminating the rigidity of the SQL schemas (Boicea *et al*., 2012).

Example of the Document-oriented database: MongoDB

“MongoDB is a cross-platform, document oriented database that provides high performance, high availability and easy scalabilty” (Stanescu *et al*., 2016). Cure *et al*., (2012) defined MongoDB as “an open-source, schema-free, document-oriented database using a collection oriented storage”. The collections in MongoDB are equivalent to tables in a relational database. Each collection contains documents that can be nested in complex hierarchies and still be easy to query and index. A document is a set of fields, each one being a key-value pair. According to Boicea *et al*., (2012), MongoDB is the most widely used document-store database and this is due to the open source nature of the system and also it is written in C++ programming language. MongoDB has been identified to be read-intensive which means it performs more read operations than it performs write operation. As stated by Cure *et al*., (2012), MongoDB provides efficient storage of binary data including large objects (e.g. photos and videos). MongoDB also provides support for indexes and object queries for fetching data. MongoDB applies scaling of reads by the use of replica sets and it scales write operations by application of sharding. MongoDB tolerates incomplete data but is not very flexible with querying as it does not offer join operations between collections (Cure *et al*., 2012).

2.5.1 Features of NoSQL Systems

"Despite operating on a significant amount of data, the core attribute of NoSQL databases is that it is able to offer such services effectively while keeping the data integrity intact" (Zafar *et al*., 2016).

Map-reduction

Zafar *et al*., (2016) defined map reduce as "a method which can process large amount of data with concurrent datasets". The NoSQL database allows Mapreduce to be used in querying the database directly as a form of command. According to Zafar *et al*., (2016), in RDBMS, the map reduce functionality requires transferring data from one database to other database but the NoSQL database can relieve this migration issue as the map reduce operation is embedded in the NoSQL system which helps in avoiding data migration from one database to another. As described by Nyati *et al*., (2013), the NoSQL databases perform the map-reduce function by first using the map function to process a key/value pair which splits the key/value pair into different sets of values, and then the reduce function is used to integrate the different sets of values which are related with the divided key.

As stated by Zafar *et al*., (2016), the data typically lies within the pattern characterized by three steps: Mapper, Shard and Reducer. This map-reduce is performed using algorithm which is suitable to the type of database involved. The Mapper part comprises the map process, which concurrently processes multiple parts of data and yields a couple, (key-value). The shard process guarantees that the mapper process is concluded. Once the mapping is accomplished, the shard sorts out the key-value pair and the reducer produces outcomes by combining the values and increasing the count after obtaining the separate (key-value) pairs (Zafar *et al.*, 2016). As opined by Grolinger *et al.*, (2014) map-reduce has been a good option in processing large volumes of data as it offers symmetric execution on a sizeable number of computing connections with the reduce function performing aggregation of the information which was delivered from the map function. This offers increased scalability and independence from the database and the programming language. Figure 2.5 shows an example of the mapreduce function and how aggregation is performed within a mapreduce-compliant system.

Map(k1,v1) = list(k2,v2) 2.1

Reduce (k2,list(v2)) = list(v3) 2.2

List : (a;2)(a;4)(b;4)(c;5)(b;2)(a;1) 2.3

After mapping : (a;[2,4,1]),(b;[4,2]),(c[5]) 2.4

After reducing : (a;7), (b;6), (c;5) 2.5

Equation for map reduction

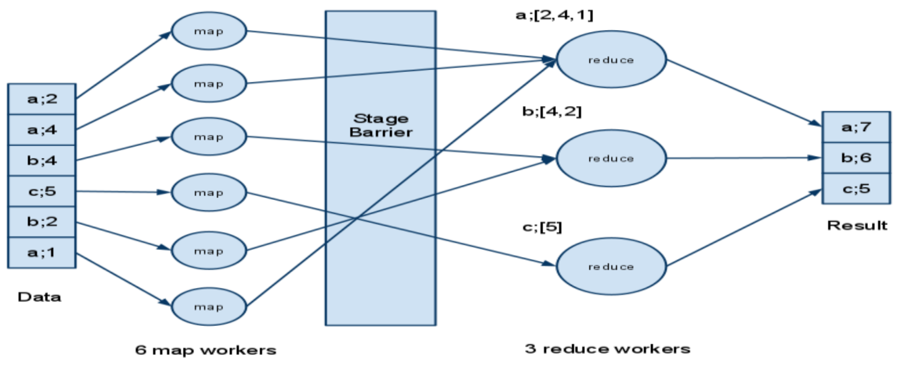


Figure 2.5 Mapreduce function (Source: Bonnet *et al.*, 2011).

High Availability and Scalability: According to Benefico *et al.*, (2012), availability is the capacity of the system to keep operating even when failure has been experienced. As a result of the distributed nature of the NoSQL database across many nodes, when there is failure in one node, due to data replication across these nodes, the system keeps functioning. NoSQL databases are unique for their provision of availability. It prioritizes availability of data over the consistency of the database.

Scalability has been identified as one of the major advantages of NoSQL systems over the relational model as data could be easily spread across many servers using high-end systems and this is due to the schema-less nature of the NoSQL systems which makes the use of joins needless (Leavitt, 2010). Join operations which are synonymous with the relational databases are very costly to perform and as Leavitt (2010) further noted, there is difficulty in performing joins on tables which are spread across a distributed system in a relational model.

Flexibility and Performance: NoSQL systems offer very high performance as the database can perform read/write operations very fast (Bonnet *et al*., 2011). Bonnet *et al*., (2011) further noted that NoSQL databases perform better in processing data across a distributed layout and can easily aggregate and update data. As argued by Li and Manoharan (2013), not all NoSQL systems execute faster than the SQL databases in terms of read/write operations but due to their ability to give maximum output while holding huge volumes of data and their flexible schema nature makes NoSQL systems unique.

Sharding: Sharding is the introduction of new node (s) to the current system which will not affect the system performance or shutdown the system but increases the capability of the system whereby data is distributed over the node and is arranged in a non-overlapped form. These nodes can be operated on different systems and as such, each node can independently eliminate source dispute (Zafar *et al*., 2016). Sharding increases the performance of the database as it helps balance out load when there is a rapid increase in calls of querying to be performed by the database which depends in part on configuration and to a large extent on horizontal partitioning of the system (Gudivada *et al*., 2014).

As shown in Figure 2.6 the shard operation introduces new nodes and data is distributed across these different nodes. Sharding offers a more robust system as the system is built to handle distribution of growing amount of data.

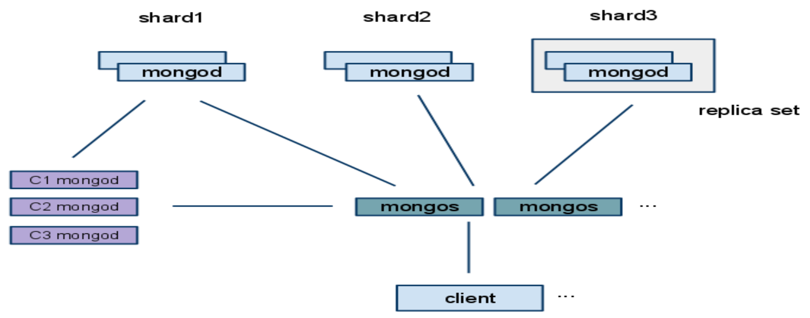


Figure 2.6 Sharding function (source: Bonnet *et al*., 2011).

Replication: Hecht and Jablonski (2011) acknowledged that NoSQL databases offer automatic data replication which means that in case of any temporary failure across a node in the database, the entire system will maintain availability and effective load balancing with replacement of that node by replica servers while the database performs recovery of the failed node. As described by Bonnet *et al*., (2011), some NoSQL systems like Cassandra use a duplication system known as Multi-Version Concurrency Control (MVCC) which stores replicated versions of the same data and matches the various versions before merging them. The MVCC is most useful when applied to a distributed system as it avoids locks and handles concurrent write operations efficiently by providing eventual consistency. Bonnet *et al*., (2011) further noted that there is an inconvenience attributable to MVCC model of replication as it has to delete old entries from time to time which lead to loss of time in the overall database throughput. Another form of replication is the Master/Slave where one server which is considered the master performs read/write operations and another server (slave) replicates data and handles read and backup operations. This form of replication is used mainly for NoSQL systems like Mongo DB which uses locks on the database (Bonnet *et al*., 2011).As some document-store NoSQL databases like MongoDB perform more read operations, Wei-Ping *et al*., (2011) stated that they make use of replica sets which allows for a greater number of slaves while only one server which is write intensive carries out write operations for the management system.

Low Latency: Boicea *et al*., (2012) noted that the NoSQL model is designed to conform to low latency rate within the management system. According to Indrawan-Santiago, (2012), due to the replication of data in NoSQL, a failure goes unnoticed as replica servers generate the exact information on the failed node. In contrast "SQL (relational database) maintains consistency through high latency (reduced response time) within the DBMS" (Wu *et al*., 2017).

As posited by Sundhara *et al*., (2017) “NoSQL databases can be used with applications that have; large transaction volumes, have the need for low-latency access to massive datasets, and have the need for nearly perfect service availability while operating in an unreliable environment”. As depicted in Figure 2.7, the NoSQL system considers when there is partition, availability and consistency of the database and when the system performs normally without partition, it considers latency and consistency.

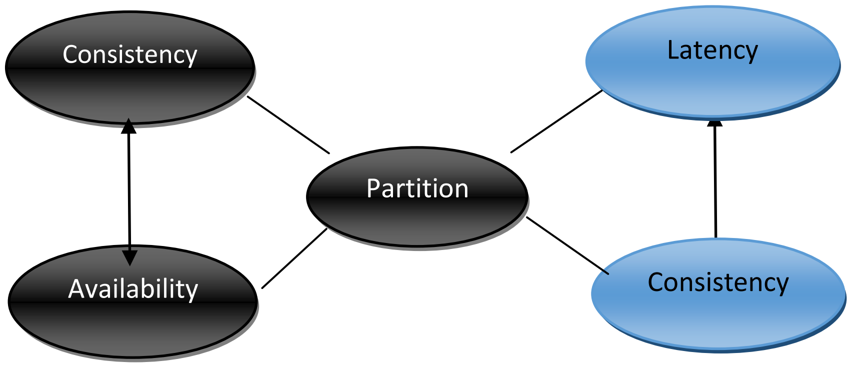


Figure 2.7 PACELC model

As inferred by Indrawan-Santiago, (2012), the PACELC (Partition, Availability and Consistency, Else, Latency and Consistency) theorem inculcates all the characteristics of the NoSQL management system. These features are interwoven in their functionality and are meant to arrive finally at a highly available database. Mapreduce produces scalability for the database, scalability allows for improved performance as data is processed with ease and speed. Sharding allows for replication where data is reproduced and spread across many servers which is based mainly on the schema-free (flexible) nature of the NoSQL systems and all these measures are put in place to ensure provision of a database that is considered to be reasonably consistent and maximally available.

2.6 Limitations of NoSQL Systems

Heterogeneous data structure

“NoSQL data stores are commonly schema-less, providing no means for globally defining the schema. While this offers great flexibility in early stages of application development, developers soon can experience the heavy burden of dealing with increasingly heterogeneous data”

(Scherzinger *et al*., 2014). Unrelated data such as emails, multimedia and blogs are stored in a schema-less table which can come from different sources for example the social media and this means varied structures of data are held together in one table. As the table grows or expands, it becomes an issue for the database to fully provide a uniform application interface which can encompass these diverse data combination (Jayathilake *et al*., 2012).

The heterogeneity is as a result of different data models which are used by different NoSQL data-stores. Even when the data model is the same, there are variations due to different implementations. There is also the issue of the differences in query language used and the type of consistency model used by a particular NoSQL Database Management System. Different NoSQL models apply different CAP formation and the combination by one NoSQL management system might not be supported by another model (Dharmasiri and Goonetillake, 2013).

Near inconsistency of the database

As noted by Han *et al*., (2011), Professor Eric Brewer in 2000 proposed the CAP theorem which stands for (Consistency, Availability and Partition tolerance). According to Bonnet *et al*., 2011), the CAP model grants that in a joint-data scheme, only two out of the three features can be satisfied at a particular point in time within the database. As they further explained, there are three possible configurations which are; consistency and partition tolerance, availability and partition tolerance and the last consistency and availability which is very difficult to combine.

In NoSQL system, availability and partition tolerance are ranked higher and valued over consistency. The system is satisfied with eventual consistency (Wang and Tang, 2012).

Also the BASE (Basically Available, Soft state, eventually consistent) pattern of the NoSQL databases proves that NoSQL systems are more disposed towards availability of data than they are towards consistency of the data involved (Gudivada *et al*, 2014).

Varying Query model: Unlike the relational model which uses the SQL (Structured Query Language) as its unified language for querying RDBMS (Relational Database Management Systems), different NoSQL vendors have different languages which they use to access their database, there is no standard query interface for the NoSQL database management systems (Dharmasiri and Goonetilake, 2013). According to Lomotey and Deters (2014), this variability in data format leaves an overhead as a result of different API which would be used in analysis of data.

2.7 Security Issues in Design Model of NoSQL Systems

According to Zahid *et al*., (2014), NoSQL systems with their inherent auto-sharding for reliable high performance and good load balancing were not ab initio designed with security as one of the key features. Sharding as they further noted pose security risks as unencrypted data is replicated and distributed across many servers in different locations allowing for vulnerability of data as unauthorized users can gain access to information. This breach could also be as a result of communications within a network that is not very secure.

Okman *et al*., (2011) stated that amongst the features of the NoSQL systems is the lack of referential integrity which is maintaining integrity constraints for the foreign key and also very little support for security at the database level. This lack of support for security within the management system means there could be breaches to data which has been stored in the database.

Amongst the incumbent problems of the NoSQL databases is the prevalence of Denial of service problem (Okman *et al*., 2011). Attackers gain access to IP addresses of users by sniffing the network and divert resources to pseudo connections thereby denying legitimate users the service allotted to them.

Security enhancement for NoSQL database: AS proposed by Zahid *et al*., (2014), some techniques also identified as loopholes such as authentication, access controls, secure configuration, data encryption and auditing can be improved upon for a secure NoSQL database system.

Authentication: Authentication is the verification of identity of users to ensure that the rightful users are granted access to database resources. Authentication can be for a single user or group access or verification between servers. To secure the sharded NoSQL database using authentication, Zahid *et al*., (2014) recommended some authentication methods like use of protocols such as SSL (Secure Socket Layer) and SSH (Secure Shell) which are efficient cryptographic models. Also, certificate based authentication where every certificate is verified and Password based authentication methods could be used.

Access Controls: This is a process for ensuring that only authorized users gain access to database resources. Some proposed access control techniques by Zahid *et al*., (2014) are RBAC (Role Based Access Control), DAC (Discretionary Access Control) and also MAC (Mandatory Access Control). These various models listed can be applied based on system configurations to ensure restricted access only for functions specified for a particular user.

Secure Configuration: Faulty configurations at the Operating System, in the database or the application layer can lead to breach of security through the entire database. The suggested configuration runs from backup to updates and services. Proper configuration of ports, files and directories and protocols was also recommended.

Encryption of data: Encryption of data is used to provide secrecy of information within the database (data-at-rest) and across the network (data-in-transit) using mathematical algorithms. Some cryptographic standards proposed are DES (Data Encryption Standard), AES (Advanced Encryption Standard) and could be used to protect data within the database. Some techniques for securing data which is sent across the network are IPSec, SSL, TLS and SSH.

Auditing: Audit trails which can be used to monitor activities performed in the database can be used to enhance security of data in the database. Auditing stands for monitoring and noting activities carried out by database users. It can be used to detect infiltration attempts by attackers. This is periodic check on connections and activities.

Summary of the various characteristics of the different NoSQL databases which have been considered for use by the developer is placed in Table 2.8. For querying in the NoSQL systems outlined, various APIs are used to achieve this purpose as there is no unified query language for NoSQL systems. The relational model has a unified query Language which is SQL and also allows for query optimization.

Table 2.8 Summary of the features of Prevalent NoSQL Databases

Features	**Redis**	**Cassandra**	**MongoDB**	**CouchDB**	**RavenDB**	**Neo4j**
Language Written in:	C	Java	C++	Earlang	C#	Java
Storage Capacity	Physical storage in Disk (small storage capacity)	Memory (large capacity)	Memory (large capacity)	Memory (large capacity)	Memory (large capacity)	Memory (large capacity)
Data model	Key-value	Column	Document	Document	Document	Graph
Sharding (horizontal Partitioning)	Not supported	Performs sharding	Performs sharding	Not supported	Performs sharding	Not supported
License type (source)	BSD Open source	Apache Open source	AGPL Open source	Apache Open source	AGPL Open source Commercial	AGPL GPL Open source
Fast Concurrent read/write	Fast concurrent read/write Read-intensive	Fast concurrent read/write Write-intensive	Fast concurrent read/write Read-intensive	Fast concurrent read/write Read-intensive	Fast concurrent read/write Read-intensive	Read/write intensive
CAP Theorem (Transaction)	Availability, Partition-Tolerance	Availability, Partition-Tolerance	Availability, Partition-Tolerance	Availability, Partition-Tolerance	ACID	Consistency, Availability
Consistency Model	Strong consistency	Eventual consistency	Strong Consistency	Eventual consistency	Strong Consistency	Strong Consistency

Features	**Redis**	**Cassandra**	**MongoDB**	**CouchDB**	**RavenDB**	**Neo4j**
Indexes (secondary indexes)	Global primary indexes(no secondary indexes)	No indexes	Has indexes	Has indexes	Has indexes	No indexes
Ad-hoc Query	None	HIVE, PIG	BSON based format (MongoSQL)	Lucene, Cloudant	Limited, built-in (JSON format)	Cypher

2.8 Review of Related Works

Some researchers have carried out extensive but not exhaustive research both academic and business on the interoperability/integration of SQL and NoSQL databases. These works are outlined in this section and analyzed according to their similarities to the proposed work.

2.8.1 Middleware Design for Integrating Relational Database and NoSQL Based on Data Dictionary

Zhang *et al*., (2011) proposed integration of relational model and NoSQL system with the help of data dictionary. This experiment is based on integration of relational databases and NoSQL approach by affixing middleware, which affords a unified interface processing across various databases. The design integrating relational and NoSQL architecture comprises three functional phases: middleware client, middleware server and SQL engine. The middleware client serves as the communication path between the middleware server and the application. The middleware client also offers user/password encryption.

The middleware server provides a distributed data transmission service, protecting dispersed and heterogeneous data sources which bring disturbance to the application cluster thereby presenting transparency in data access. The middleware server receives requests from the application, analyzes it then transmits the request to the principal database.

As shown in Figure 2.8, the SQL engine translates standard SQL requests into different types of access to the NoSQL database as the NoSQL products do not offer a standard SQL interface but uses the help of the internal API (Application Programming Interface) to access their data. The SQL engine serves as a wrapper integrated into NoSQL cluster's individual nodes (Zhang *et al*., 2011).

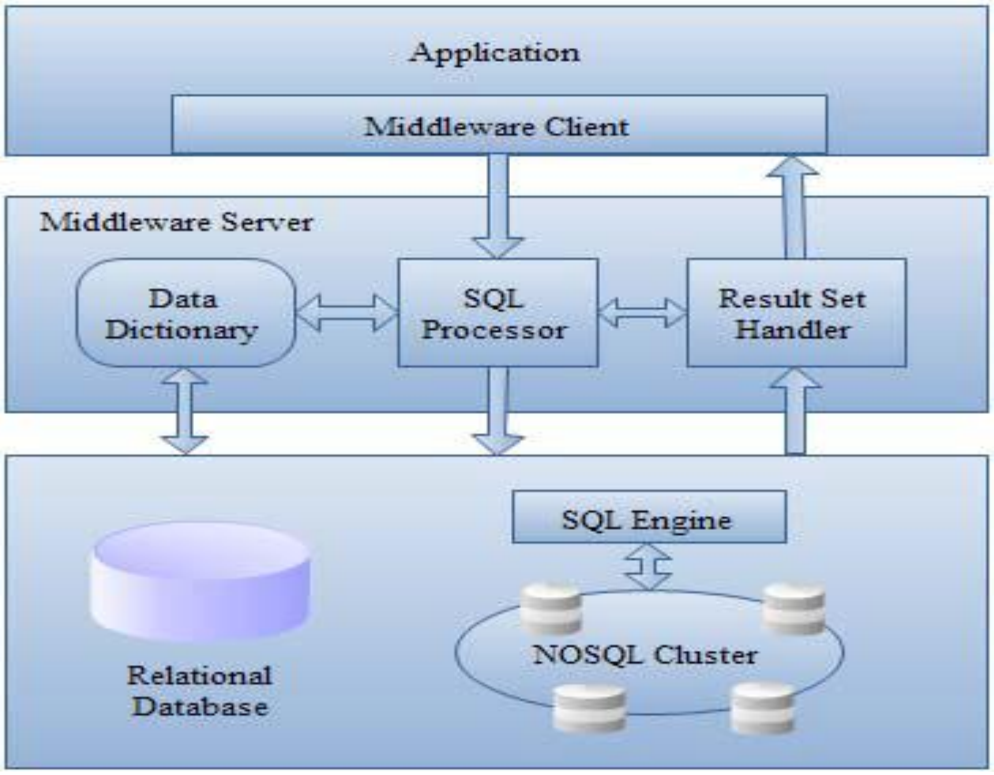


Figure 2.8 Database integration architecture (Zhang *et al*., 2011).

Zhang *et al*., (2011), designed two key technologies for the experiment for the integration of column-oriented NoSQL system (Cassandra) and relational model (Oracle Database) which are the data dictionary design and the SQL processor and Result Set Handler Design. The Data dictionary holds the routing information as well as node information which describe each database source and this enables communication with various types of databases. The SQL processor and Result Set Handler perform certain steps of the traditional SQL Engine for example Input: SQL requests such as update, delete, insert, select and call procedure are inputted by the SQL processor. Parse SQL: the SQL Processor converts the part of the SQL statement into internal data structure while the Result Set Handler integrates the results returned from numerous data sources.

The researchers noted that more work is required in terms of database integration which is as a result of the heterogeneity and distribution of data. This is because NoSQL which supports horizontal scaling cannot handle complex SQL requests.

2.8.2 Graph Data Warehouse: Steps to Integrating Graph Databases into the Traditional Conceptual Structure of a Data Warehouse

As has been highlighted in the background study of the first chapter of this work, Liu and Vitolo (2013) worked on graph data warehousing with emphasis on eliciting the steps to integrating graph databases into the traditional conceptual structure. This research explained that the concept of 'Graph Cube' is proposed as a design which incorporates graphs with tables. This model functions as a prototype which is the basis of a graph data warehouse. Some DML (select, insert, update and delete) and DDL (create, drop and update) in SQL is synchronized with the graph data model to give GDML (Graph Data Manipulation Language) and GDDL (Graph Data Definition Language). This model considered and executed the use of views in the graph data warehouse using the concept "graph view" (GVIEW) which is a logical replica of graph data, residing in local libraries. The GVIEW was implemented to meet two main purposes which are;

1. *To deliver a local copy of data*. With GVIEW, operations on the database are safer and faster as the GVIEW permits synchronized multiple usage of the graph on individualized perspectives of the database.
2. *To provide a depiction of a graph or graph segment*. GVIEW is an appearance of the graph data on the server. Added information such as, the locations of each node from the last alteration is saved in GVIEW. High-end users whose work depend heavily on direct observation will find this feature attractive, as it avails a convenient way to "save" or "remember" last visualized graphs.

The experiment started with the development of a Java-based application programming Interface (API) accessing a Neo4j graph database. Some functionalities of the relational database were initiated with the GDML and GDDL such as: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP etc. which uses a graphical user interface (GUI) to provide a more direct and responsive visualization of the data in the graph.

The graph and relational models as explained by Liu and Vitolo (2013) are fundamentally different in their methodology; while the GDBMS (graph database management system) employs physical adjacency, the RDBMS (relational database management system) employs synthetic adjacency inferring a relationship due to the key-field for instance.

The GCUBE concept is basically the incorporation of graphs as part of fact tables which will when finally implemented have a “join” function which leverages two characteristics; inclusion of graph data into the mature data warehouse technologies and enhancement of the capabilities of data warehouse with practical abilities of graph data. The researchers suggested that future work be carried out on projects which incorporate join-operations between relational model and graph model. The core objective will be to connect relational data to graph data and provide an aggregation/connectedness for multiple discrete data.

2.8.3 Blending SQL and NewSQL Approaches

Doshi *et al*., (2013) focused on combining SQL and NewSQL approaches. Their methodology was based on handling data growth which have been previously mentioned in the work and classified into three categories; chronological (accumulation of data over days, months or years), horizontal (growth in data as a result of new services or progressions introduced in an organization to solve new business needs) and vertical (rise in business scale and intricacy). This architecture looked at the various storage needs and blended the appropriate relational and NewSQL/NoSQL system to achieve an optimal storage system that leverages prolific data management. Various approaches were employed: reference architectures describing the various data handling techniques. The research used two software utilities; Instant Data Integrator (IDI) and Hibernate OGM-HBase Driver.

The instant data integrator as explained by Doshi *et al*., (2013), is also known as SQL-to-NewSQL sync. The integrator provides a streaming “Extract-Transform-Load (ETL)” proficiency for reproducing changed data from a SQL database into a NewSQL cluster almost rapidly. This is achieved by parsing the redo log to pull out changes that get committed on the SQL side of the architecture. The changes which have been committed are then converted into a target-appropriate schema and then loaded into the NewSQL cluster. The Instant Data Integrator is made up of three layers; Extractor Layer, Transformer Layer and Loader Layer.

The extractor layer implemented extractor plug-ins for IBM DB2 and Oracle Databases which are modules for each prominent SQL database. The plug-ins is used to observe redo logs and ascertain the data changes based on the log design specification.

The redo log on the SQL side aids the instant data integrator to eliminate the need for a developer to write difficult or bulky queries against large datasets in order to find the fairly small changes over a given time interval.

The transformer layer handles extracted data which are transformed again in the transformer layer using a set of plug-ins which are reposed in the transformer layer. This plug-ins is known as "Direct-Map plug-in" which is adequate when transformed data does not have to go through a format transformation.

The final phase of the IDI is the loading process. This is achieved through provisioning of a NewSQL interface whereby the instant data integrator provides a modular load function in its loader layer. The loading process identifies incremental changes on one side and reflects them in the other side.

The whole process of the IDI which spans the extraction, transformation and loading plug-ins is geared towards achieving accommodating varying requirements and constraints and to approach almost real-time reflection of SQL sides updates into the NewSQL cluster" (Doshi *et al*., 2013).

Hibernate OGM-HBase Driver & ORM-Hive (Object Relational Mapping -Hive) Dialect

The second tool used by the Unity architecture is the Hibernate OGM (Object Grid Mapping) which consists of two packages that combine to allow for ease of use of common data access logic for SQL and NewSQL repositories. One of the packages is the Hibernate OGM family which includes HBase and this is known as the Hibernate OGM HBase driver; the second package is a utility that adds Hive dialect to Hibernate which uses JPA-QL (Java Persistence API – Query Language) to translate naturally to Hive-QL.

The basic function of the Hibernate framework and utilities is to provide mapping of data operations from Java's object-oriented domain to the logical row-column model of a relational database domain. The Hibernate libraries handle transformations required for persisting/inherent objects and their associations/dependencies with one another and for searching and retrieving them automatically into/out of a relational model. The Hibernate libraries also take care of many commercial and open source databases through necessary SQL dialects.

The Hibernate framework frees application developers from the burden of semantic mismatch (incompatibility between the database and the programming language) between the two domains which enables the business logic to remain orderly and manageable.

The Hibernate framework uses an application for carrying out querying known as Hibernate Search for retrieval and searching the backend store. Hibernate OGM inculcates the Hibernate concept through to the case where a data grid delivers a cluster caching capability in front of a data store thereby enabling a natural extension to NewSQL repositories.

In conclusion, Doshi *et al*., (2013) proposed future work to be done in the aspect of "blending SQL IMDBs (Internet Movie Database) with distributed, fault-tolerant, memory-centric NewSQL stores including such systems as the Shark-Spark framework". The other work proposed is as a result of challenge arising in very high volume data processing need which requires very highly scalable but lower cost object storage systems which demands a combination of in-memory and massively parallel computing.

2.8.4 Integration and Virtualization of Relational SQL and NoSQL systems including MySQL and MongoDB

Lawrence (2014) proposed an integration and virtualization model for relational and NoSQL systems using MySQL and MongoDB as use cases. The research was based on a "generalizable SQL query interface for both relational and NoSQL systems called Unity". Unity model permits SQL queries to be automatically converted and implemented using the core API of the data sources (NoSQL or relational). Unity is basically integration and virtualization system which agrees to SQL queries that extend to numerous sources and uses its interior query engine to implement joins across sources. The key features of Unity are outlined below:

1) A SQL query processor and optimizer including support for push down filters and cross-source hash joins.
2) A customizable SQL dialect translator that uses mappings to compensate for different SQL variants in relational systems.

The Unity architecture according to Lawrence (2014) is an integration and virtualization system that allows SQL queries across relational and NoSQL systems. The virtualization/execution layer allows converting SQL queries to NoSQL APIs and automatically performs operations not supported by NoSQL systems. Unity allows NoSQL systems to effortlessly interact with relational databases and enterprise reporting applications but the downside to the experimental results from Unity architecture is the minimal overhead in the SQL translation process. Future work was proposed by Lawrence (2014) in terms of increasing the performance of other supported NoSQL systems such as Cassandra and also in terms of "parallelizing the virtualization/execution engine for a cluster environment".

2.8.5 Schema Conversion Model of SQL Database to NoSQL

Zhao *et al*., (2014) developed a method for converting SQL schema to NoSQL. The researchers proposed a general schema conversion model for converting relational database to NoSQL database which help migrating to NoSQL and improves reading efficiency. The schema conversion scheme technique applied the idea of table nesting to improve the performance of cross table query. The idea of table nesting certified that there must be a one-to-one mapping between tables in relational database and dataset in NoSQL. An example is the MongoDB; if there is n number of tables in relational database, there must be corresponding n number of collections in MongoDB. This procedure ensured that when storing structured data with references in NoSQL database, the references are considered as relationship between parent layer and children layer of semi-structured data in NoSQL database. The idea of nesting is that storing referred (referential) tables as children tags in data item of the referring (primary) table. This process enables semi-structured data with multiple layers to define structured data with multiple references between tables, which covers all situations of table references.

This approach relies on the following orientations: denormalization, migrating data to NoSQL database, and converting schema to NoSQL database.

For data migration, the researchers employed two key technologies, Apache Sqoop and DataX. The Apache Sqoop serves to transfer bulk data between Apache Hadoop and the structured data storage such as relational database. These tools can import data from the relational model into HDFS (Hadoop Distributed File System), convert the data in Hadoop MapReduce, then pass on the data back into a relational model. According to Zhao *et al*., (2014) the limitation of the Apache Sqoop is that it is not able to import the dependencies of tables in relational database, even though it can complete data importation in HBase, Hive and Hadoop. Due to this downside, there is a lot of overhead cost in terms of time spent during data migration from relational to any big data management system such as Hadoop, Hive or HBase as the technology requires more than one table to be accessed and queried.

DataX is a technology designed to leverage very rapid data communication/exchange between heterogeneous data sources for example; MYSQL, Oracle and HDFS. DataX uses its DataX engine to deliver a framework to schedule, process and exchange data. The activities of data extraction and loading are handled by DataX plugins which are plugged into DataX engine. The DataX technology also has the limitation of inability for importing relationship of tables in relational database to NoSQL database just like Apache Sqoop.

For the schema conversion, Zhao *et al*., (2014) used JackHare, a framework which translates data from SQL to NoSQL using MapReduce by generating nested schema mappings which transforms relational data into HBase representation automatically. The JackHare delivers a concrete conversion model which caches all tables in a relational database in a single HBase table. In the conversion process, the data contained in an SQL table are transferred into a column family and column schema of the relational table is remapped into qualifier of that column family. In addition to the conversion process, a special column family is adopted for foreign keys of relational database. The limitation of this foreign key introduction is that query involving many foreign keys will not have optimum throughput which is as a result of the join operations to be processed.

In conclusion, the research achieved a schema conversion which used nested idea to increase query speed of NoSQL database. The research proposed a broad migration program and associated algorithms based on the nested method. The experiment accomplished a MySQL-MongoDB relocation system, and employed real-world data to ascertain the correctness of migration and query performance which were compared. Also, conversion algorithms were tested for accuracy. The weakness of this model is the overhead cost of a certain amount of space which is needed in exchange for query efficiency. For the reason of the downside, the researchers proposed future work which explores ways for minimization of spatial redundancy.

2.8.6 Cross Platform (RDBMS to NoSQL) Database Validation Tool using Bloom Filter

Goyal *et al*., (2016) worked on creating a cross-platform relational model to NoSQL model using bloom filter. The researchers explained that the concept of data migration from relational database to NoSQL are divided into two general approaches: 1) Direct mapping: whereby data from the source is transformed into the data structures of the target database, without considerable alteration to the schema 2) Intermediate Mapping: that is, data is translated into transitional structure and then from this into the final setup. The intermediate mapping approach is typically a denormalized schema which comprises of joins and skipping some columns in relational database model.

Goyal *et al*.,(2016) compared some existing tools which have been used for data integration/migration for example: Pentaho Data Integration (a client ETL application which supports NoSQL platforms such as MongoDB and HBase). The Pentaho Data Integration allows execution of ETL jobs and grants access and retrieval of data in big data environment such as Hadoop and also allows the use of user-defined validation rules such as tolerating null values, decimal symbols, and maximum string to validate data in the course of the transformation process from the source to the target database. Talend Open Studio was also compared. Talend Open Studio as described by Goyal *et al*., (2016) is an ETL tool which supports different NoSQL databases such as MongoDB, CouchDB, CouchBase, Cassandra and HBase and offers the functionality to link and integrate them. Jaspersoft ETL, a tool circulated by TIBCO which supports NoSQL platforms such as Cassandra and MongoDB which in partnership with Talend allows the use of Talend Open Studio in its environment was also compared by the researchers.

Figure 2.9 explains the flow of algorithm used in the experiment for migration/integration of the RDBMS and NoSQL by (Goyal *et al*., 2016). The research projected a comprehensive, fast and space proficient algorithm to authenticate RDBMS to NoSQL migration. The experiment accomplished desirable values when tested with different sizes of test data.

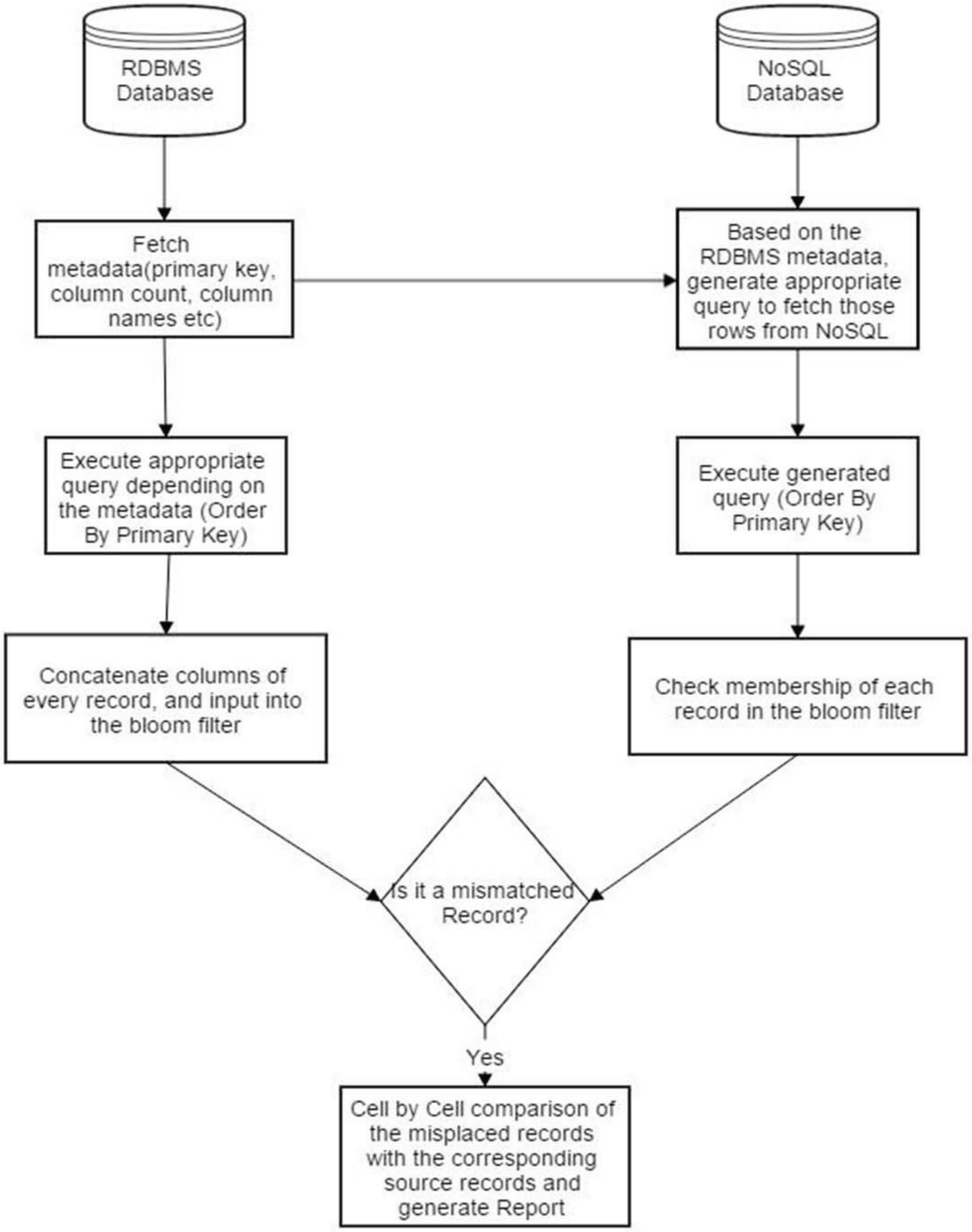

RDBMS Database
NoSQL Database
Fetch metadata(primary key, column count, column names etc)
Based on the RDBMS metadata, generate appropriate query to fetch those rows from NoSQL
Execute appropriate query depending on the metadata (Order By Primary Key)
Execute generated query (Order By Primary Key)
Concatenate columns of every record, and input into the bloom filter
Check membership of each record in the bloom filter
Is it a mismatched Record?
Yes
Cell by Cell comparison of the misplaced records with the corresponding source records and generate Report

Figure 2.9 Validation flow diagram (Source: Goyal *et al*., 2016)

Goyal *et al.*, (2016) used Bloom filter to checkmate this weakness in implementation of migration and integration of data from relational database to NoSQL database. Bloom filters helped correct the drawbacks by revealing the exact corrupted records for massive datasets and also achieving constant space and linear time complexity. The researchers proposed an approach which validates data conversion from relational(MySQL) to NoSQL(MongoDB and Apache Cassandra) databases and this is outlined in four phases; i) Metadata mapping from RDBMS to NoSQL database ii) Transformation into a common structure iii) Bloom filter iv) Cell validation engine.

The three methods; Pentaho Data Integration, Talend Open Studio and Jaspersoft which were examined by Goyal *et al.*, (2016) have some major weaknesses like they do not provide row and column comparisons over the entire data set thereby causing loss of data. The main limitation of this research is the small probability of false positives which can be removed by several optimizations made to the bloom filter. The authors of the paper proposed further research to their work which will be in terms of exploring "ways to parallelize the entire process and dynamically obtain bucket sizes for quicker evaluation".

This research having been examined for further research, falls short of integrating other complex databases like the key-value pair NoSQL DBMS which provides optimization for insert operations and is ACID compliant was not incorporated intop the architecture and system development. Also, the graph-oriented NoSQL model which provides specialized feature for handling graphical operations linking relationships involving images was not addressed in the existing research. The proposed research took into account these drawbacks and a persistent system offering interoperability across board for the entire NoSQL system and the SQL model was built.

2.8.7 Automatic Mapping of MySQL Databases to NoSQL MongoDB

Stanescu *et al.*, (2016) worked on MySQL and NoSQL integration by performing automatic mapping of the MySQL relations/tables to the equivalent NoSQL documents by the help of a middle layer. The research presented a framework that implements a novel algorithm that automatically maps a relational database to a MongoDB NoSQL database. The algorithm makes use of the metadata (data about data; for example the name of a database or table, the data type of a column, or access privileges) deposited in the MySQL system tables which considers mainly the idea of the ER (Entity-Relationship) model which is a concept chiefly associated with the relational model. The mapping uses the 1:1 (one to one) and 1:M (one to many) relationship type corresponding to the Foreign Keys in the relational model and N:M (many to many) relationship type represented in the relational model with a join table that contains the Primary Keys (PK) from the original tables (MySQL tables) and the join table using the MySQL INORMATION_SCHEMA (the information database; a repository where all the information about all the other databases that MySQL server maintains are stored) that provides access to database metadata. Within the INFORMATION_SCHEMA, there are several read-only tables which in essence are views (virtual representation of the data in a table), not base tables.

The 1:1 relationship depicts a relationship between two entities. For example a student has a single Address relationship which basically means that a student lives a single Address and an Address only contains a single Student. The 1:1 relationship in the automatic mapping framework deduced by this research offers two modeling techniques using MongoDB; the first is to embed the relationship as a document and the second is to link to a document in a separate collection. The embedding technique which is a core attribute of the MongoDB is the chosen method for modeling the relationship because it is more efficient in retrieving the document. The 1:M (one to many) relationship illustrates a relationship where side has a single relationship and the reverse side can bear more than one relationship. The 1:M relationship can be represented in numerous ways using MongoDB just like that of the 1:1 relationship and these modeling formulations include embedding, linking and bucketing strategy which does not apply in the case of 1:1 relationship. Bucketing strategy is chiefly employed in the case of time series analysis (a statistical technique that deals with trends or time series data within a particular time periods or intervals). The N:M (many to many) relationship in the ER (Entity-Relationship) model is a pattern of relationship between two entity types where both entities might possess many relationships between them. For example a book might be written by many authors, also an author might have written many books. The N:M (many to many) relationships are modeled in the RDBMS by using a join table containing the primary keys from the original tables, each representing a foreign key, and two 1:M (one to many) relationships. This N:M relationship of the relational database can be represented in MongoDB in two different forms which are; One Way Embedding Strategy and Two Way Embedding Strategy. In the One Way Embedding strategy of the automatic mapping of MySQL tables to MongoDB, the strategy opts to optimize the read performance of a N:M relationship by embedding the references in one side of the relationship. For example a N:M relationship between books and categories, several books belong to a few categories but a couple of categories can have many books. The Two Way Embedding offers double embedding in both sides of the relationship for the author of book and also the book by including the foreign keys for book under the book field in the author's document. This is made possible by mirroring the author document for each book and including the author's foreign keys under the author field in the book document. An example with the categories represented into a separate document is shown below;

An example of Category documents: {id=1,
Cname=”Multimedia”}
{id=2,
Cname=”Databases”}
An example of a Book document with foreign keys for Categories:
{id:1,
title”Multimedia Databases”,
categories:[1, 2],
authors:[1]}
id:2,
title”Multimedia”,
categories:[1],
authors:[1,2]}
The stepwise process of the algorithm implemented in the framework by Stanescu *et al.*, (2016) are;

1) MongoDB database creation.

The process involves specification of the MySQL database that will be represented in MongoDB. The database is created using the following MongDB command

use DATABASE_NAME

>use db1

Switched to db db1

2) The second step involves creation of tables in the new MongoDB database

The algorithm performs verification for each table to ascertain the type of relationship involved whether there are foreign keys involved or if the table is referred by other tables.

2.1 If the table created in MongoDB is not referred by other tables in the MySQL, it will be represented by a new MongoDB collection.

2.2 If the new table has not foreign keys, or has been referred by another table, it will be represented by a new MongoDB collection.

2.3 If the table has one foreign key and is referred by another table, it will be represented by a new MongoDB collection. In the automatic mapping framework, for this type of table, the linking method (an embedded document is linked to a separate collection) is employed, using the same concept of foreign key.

2.4 If the table has one foreign key but is not referred by another table, the proposed algorithm makes use of the one way embedding model. The table is embedded in the collection that represents the table from the part 1 of the relationship.

2.5 If the table has two foreign keys and is not referred by another table, it will be represented using the two way embedding model.

2.6 If the table has 3 or more foreign keys, so it is the result of a N:M ternary, quaternary relationships, the algorithm uses the linking model, with foreign keys that refer all the tables initially implied in that relationship and already represented as MongoDB collections. The solution is good even if the table is referred or not by other tables.

To find the name of the tables stored in MySQL database this Select command is used:

Select table_name From information_schema.tables

Where table_schema=’db1’ Order By table_name;

Relational tables become collections in MongoDB. The collections are created using createCollection() method. Basic syntax of createCollection() command is as follows: Db.createCollection(name, options)

Where name represents the collection name and options specify options about memory size and indexing.

Stanescu *et al*., (2016) explained that the proposed algorithm of automatic mapping a MySQL relational database to a MongoDB NoSQL database has limitations which are in terms of not being able to experiment on complex databases using many tables with large number of databases. Future work in this area will involve the afore mentioned limitation and also take into consideration the number of records in the tables and performing operations on the database which will cover the CRUD (Create, Read, Update and Delete) function in order to implement more suitable model of mapping to MongoDB. Future work will also involve modeling tree structures with parent references and extending the framework to execute mapping to MongoDB of other relational databases for example Oracle, MS SQL Server and so on.

2.8.8 SQL Support over MongoDB using Metadata

Sanobar and Vanita (2013) worked on supporting SQL system; MySQL over a NoSQL system; MongoDB with the help of Metadata. The research based majorly on integration of MySQL and MongoDB databases by addition of a middleware (Metadata) between the front-end (application layer) and the back-end system (database layer). The paper first carried out detailed analysis of the two systems (MySQL and MongoDB) in order to ascertain the gains accruable by the integration of these systems which can operate in parallel achieving the tasks which they have been built to perform. The middleware between these two consists of Metadata which consist of different types of packages. The system implemented a package which serves as an interface between java application layer and NoSQL database (MongoDB).

The researchers reviewed the architectural underlining by analyzing the different systems; MySQL and MongoDB database management systems. Sanobar and Vanita (2013) explained that relational database is extensively used in most of the application to store and retrieve data. The RDBMS work best when they handle a restricted amount of data. Handling a huge volume of data like internet was inefficient in RDBMS. In order to surmount this quandary “No SQL” came into existence. The term NoSQL is short for “Not Only SQL” and was introduced in 2009, when it was chosen a title of a conference “for folks interested in distributed structured data storage”. The NoSQL name tried to mark the emergence of a growing number of non-relational, distributed data stores that often did not attempt to provide ACID (Atomicity, Consistency, Isolation and Durability).

Sanobar and Vanita (2013) further explained that NoSQL, is not a tool, but a methodology composed of several balancing and competing tools which handle unstructured data such as documents, e-mail, multimedia and social media efficiently. Most of the common features of NoSQL database can be summarized as schema is not fixed, does not support join operations, high scalability and reliability, very simple data model, very simple query language, high availability at the price of losing the ACID trait of the traditional database

in exchange for keeping a weaker BASE (Basic Availability, Soft State, Eventual Consistency) feature, uses cheap commodity server to manage exploding data and thus leads to low cost, efficient use of distributed indexes and RAM for data storage, ability to dynamically add new attributes to data records, ability to replicate and to distribute data over many servers.

According to Sanobar and Vanita (2013) MongoDB supports indexing over embedded objects and arrays which avails it of a special feature for arrays called “multikeys”. The multikey feature allows the use of an array as index, which can be used for searching documents by their associated tags.

The database structure in Figure 2.10 shows the layout of MongoDB database where the data model is a collection of documents which are comprised of key-value collections.

Figure 2.11 depicts the architectural framework of the MongoDB NoSQL system where sharding (horizontal partitioning) is used in segmenting the data and allowing for interaction between the application layer which is the client side and the database server.

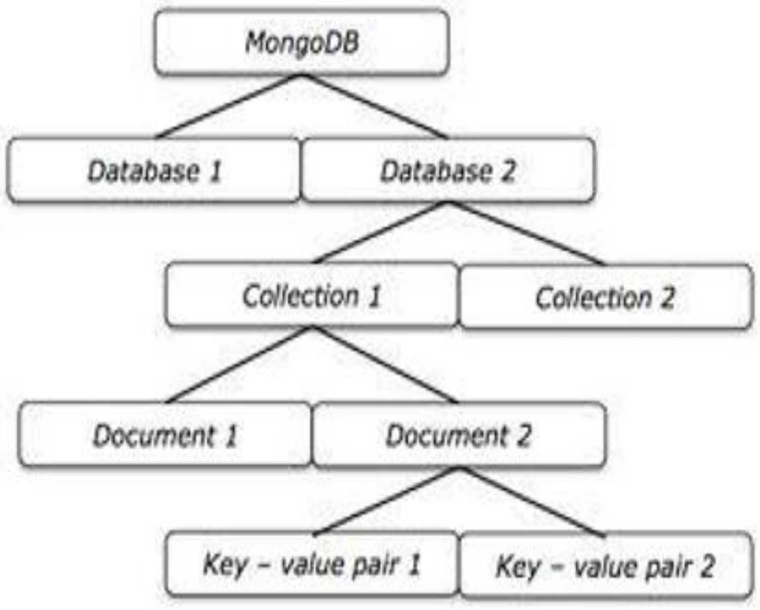


Figure 2.10 Structure of MongoDB Source: (Sanobar and Vanita, 2013).

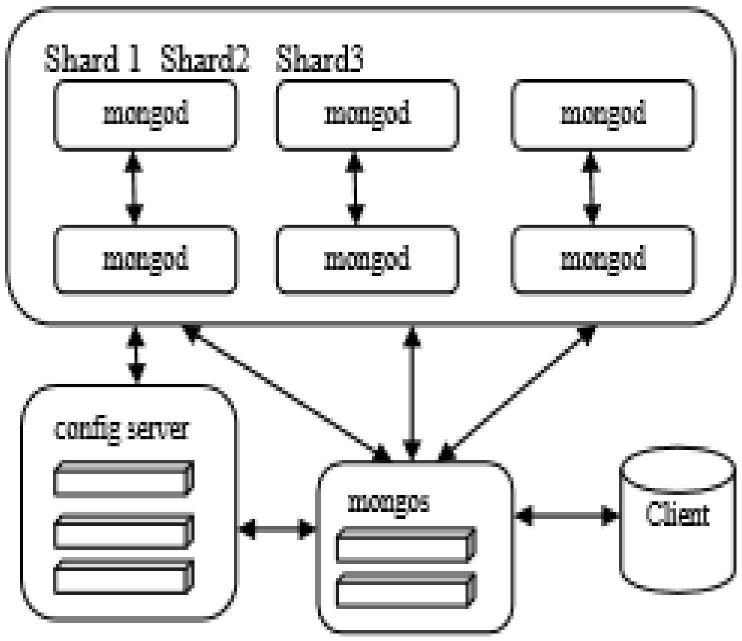


Figure 2.11 MongoDB Architecture (Source: Sanobar and Vanita, 2013).

In the implementation phase of the research carried out by Sanobar and Vanita (2013) the middleware between the MySQL and MongoDB system comprised of Metadata which consists of different packages; the package which acts as an interface between java application layer and NoSQL database (MongoDB). The system is principally designed for embedded java based application which requires database access. The database command issued from the java-based application is set as input to the interface performing as a middleware. The interface performs the parsing of the input data and reformats the code in the format which is requisite by the back-end database which is the MongoDB. The reprocessed code is made up of functions that direct back-end databases to execute and sustain the database. This process of employing a middleware helps for java response in case of an exception (error) or system failure. The Metadata (middleware) stores information about the format conversion rules and the data structure format in order to be able to perform conversion from one form to the other for both systems (MySQL and MongoDB). The drawback of this research is the lack of trial of other NoSQL systems apart from MongoDB to leverage support over other NoSQL systems to discover their performance strengths.

2.8.9 An Effective Scalable SQL Engine for NoSQL Databases

Vilaca *et al*., (2013) conducted a research which sought to offer an efficient SQL system which could be scaled to accommodate NoSQL databases. The research was carried out in order to tackle the inadequacy of existing NoSQL databases with an effectual approach for executing SQL queries while retaininging their scalability and schema flexibility. The research displayed how a full-fledged SQL engine can be integrated atop of HBase leading to an ANSI SQL compliant database. Under a standard TPC-C workload the prototype system scales linearly with the number of nodes in the system and outperforms a NoSQL TPC-C implementation optimized for HBase.

As posited by Vilaca *et al*., (2013) cloud-based databases for example; Cassandra, HBase, Google's BigTable, Amazon's DynamoDB and Yahoo's PNUTS offer Paas (Platform-as-a-service) feature and for this reason, they have become open for wider usability and adoption for superior and more diverse set of applications. The researchers further noted that their lack of support for SQL stands as a key obstacle for a wider adoption. This hurdle arises at different levels due to the dominance of SQL as the standard, widely mastered and efficient query language for databases. Most web scale applications are purposely kept SQL-based for their core data management. Any application at least two, three years old is directly or indirectly (e.g. on top of an object-relational mapping) based on an SQL interface and its migration is usually not straightforward. A huge number of tools and middleware coupled to SQL have been developed and matured over the years and are currently at the basis of most application development frameworks. For this reason, Google App Engine has requested for SQL-compliant additions to its platform which is made possible by the employment of a middleware that exposes higher-level query interfaces on top of the barebones key-value primitives of NoSQL databases.

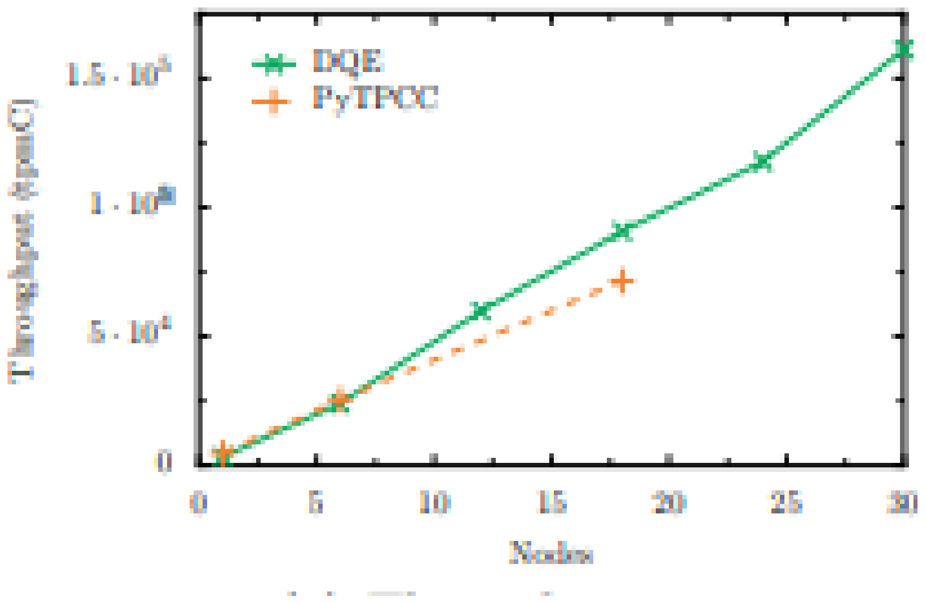
Vilaca *et al*., (2013) presented DQE (Distributed Query Engine) for running SQL queries on top of a NoSQL database, while maintaining its scalability and schema flexibility. The DQE permits combination of the expressiveness and performance of an RDBMS with the scalability and flexible schema nature of a NoSQL database. The DQE (Distributed Query Engine) was implemented using Apache Derby's query engine with full SQL support and HBase as the NoSQL database. The proposed architecture by the researchers reuses some components from the SQL query processor which are: the JDBC (Java Database Connectivity) driver; the application programming interface for the programming language Java, which defines how a client may access a database, it also uses client connection handler; the compiler and the optimizer, and a set of generic relational operator implementations.

The components of the DQE architecture to be implemented include; 1) mapping from the relational model to the data model of a NoSQL database which will include: atomic data types and their representation, representation of rows and tables, and representation of indexes. 2) Mapping from the relational schema to that of the NoSQL database, which allows data to be interpreted as relational tables 3) Implementation of sequential and index scan operators which will include: matching the interface and data representation of the database and leveraging the indexing and filtering capabilities in the NoSQL database to minimize data network traffic. As stated by Vilaca *et al*., (2013) the DQE architecture has the key advantage of remaining stateless regarding application data. DML (Data manipulation Language) statements such as; SELECT, INSERT, UPDATE and DELETE can be executed without any coordination among different DQE instances. As a consequence, the system should retain the scale-out capabilities of the supporting NoSQL database. Additionally, this architecture also leverages the possibility to take advantage of the flexible schema exposed by the underlying NoSQL database, which implies that each application applies its own view of the schema over the NoSQL database.

Vilaca *et al.,* (2013) reviewed the systems (Apache Derby and HBase) for use in the DQE architecture. The researchers defined HBase as a key-value based distributed data storage system based on Bigtable. HBase stores data in the form of HBase tables (HTable) that are multi-dimensional sorted maps. Data is maintained in dictionary ordered by row key which possesses column name and timestamp. The columns are grouped into column families which must be created before data can be stored under any column key in that family. The column can posses multiple versions of the same data indexed by their timestamp. Read/write operation is carried out on a row by the help of the row key and one or more column-keys. HBase performs the scale-out attribute of the NoSQL database management system. As explained by Vilaca *et al.*, (2013) Apache Derby holds a store layer which is divide into two main areas; access and raw. The access layer presents a combination of table or index/ row based interface to the SQL layer.

The access layer handles table scans, index scans, index lookups, indexing, sorting, and locking policies and transactions. The access layer is directly placed on top of the raw store which is the repository for rows in pages in files, transaction logging and also transaction management. For the purpose of this research, the researchers removed the raw store layer in the prototype of the DQE and also, some components of the access layer were replaced.In the prototype system, some components were employed which include; query engine, storage and file system. Applications issue SQL requests to any query engine node which in return communicates with storage nodes and executes queries which returns the results to applications. The components resident in Derby were used to implement the query engine. The query processing sub-system (the compiler and optimizer) components were re-used to help in performing join and aggregations which are not supported by the HBase. The DQE leverages indexing and filtering capabilities of the HBase to minimize the amount of data that needs to be obtained. The HBase also helps to reduce network traffic between query engine and HBase. The Apache Derby helps maintain consistency across the DQE as it must store information about the in-memory representation of tables and index in a persistent manner.

As shown in Figure 2.12, the evaluation of the DQE addressed two performance aspects which are; the overhead incurred by the prototype system in terms of added latency (reduction in response time), and throughput. The system is capable of transferring more data when more nodes are added to the system which is as a result of its ability to scale-out (increment in the number of nodes).



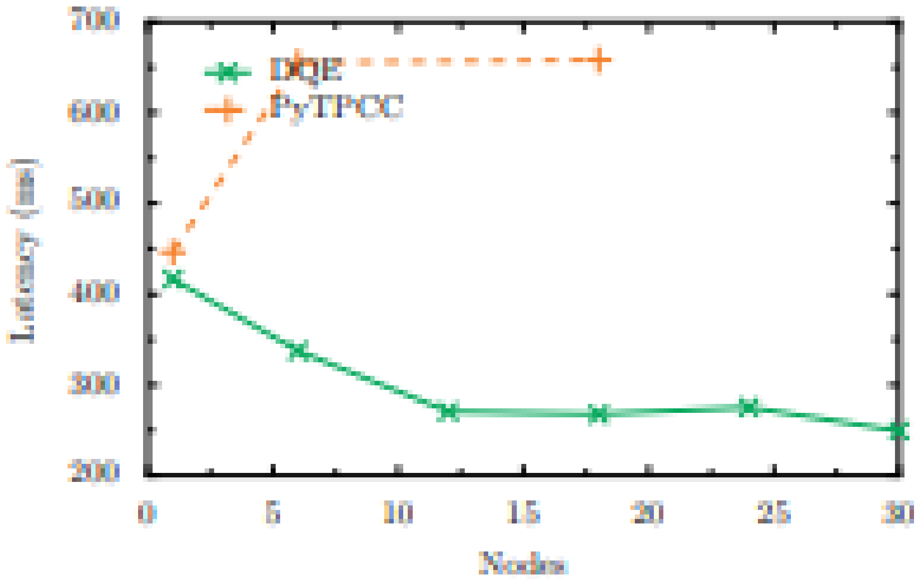


Figure 2.12 Throughput and Latency graphs for DQE (Source: Vilaca *et al*., 2013).

Vilaca *et al*., (2013) used an existing SQL implementation, without modification s to drive the DQE and an existing NoSQL implementation where columns are grouped into column families which are named differently for optimization and data storage which has been optimized for HBase. The results of the researchers' system showed that the DQE presents linear scalability which is as a result of the scale independence of the query processing layer and the scalability of the NoSQL database layer. The downside to this architecture is that the DQE has a slightly reduced throughput than the HBase but offers better scalability than the HBase NoSQL system.

Additionally, the DQE architecture incurs greater overhead operationally as a result of the manual optimization and denormalization which is caused by the Apache Derby's relational operators, filtering and secondary indexes.

2.8.10 Uniform data access platform for SQL and NoSQL database systems

Vathy-Fogarassy and Hugyák (2017) worked on implementing a uniform data access interface for MySQL relational database system and MongoDB NoSQL system. The authors designed a web-based application named HybridDB; a novel data integration methodology to query data individually from different relational and NoSQL database systems. The study employed the use of metamodel-based merging approach which allows querying of data from heterogeneous database systems simultaneously by converting the metadata (data about data) information of source database systems into general schemas which are represented in JSON (JavaScript Object Notation) objects. The research aimed to support data integration by concealing the specific details and heterogeneities of the various source systems (MySQL and MongoDB) by using a metamodel approach whereby the distinctive interfaces and schemas of the source database systems are combined into a single interface.

The proposed method by Vathy-Fogarassy and Hugyák (2017) incorporated the JSON (JavaScript Object Notation) principle which was used in mapping different data models into a common one.

As highlighted in Figure 2.13, the HybridDB model comprises of the server side and the client side architectural design. The server side was developed in Node.js JavaScript runtime environment for the reason that this platform makes use of an event-driven, non-blocking asynchronous I/O (Input/Output) model. On the server side, Vathy-Fogarassy and Hugyák (2017) applied the model-view-controller (MVC) architectural pattern. JavaScript codes on the client side were written in AngularJS framework, which is conceivably the most popular client-side JavaScript framework.

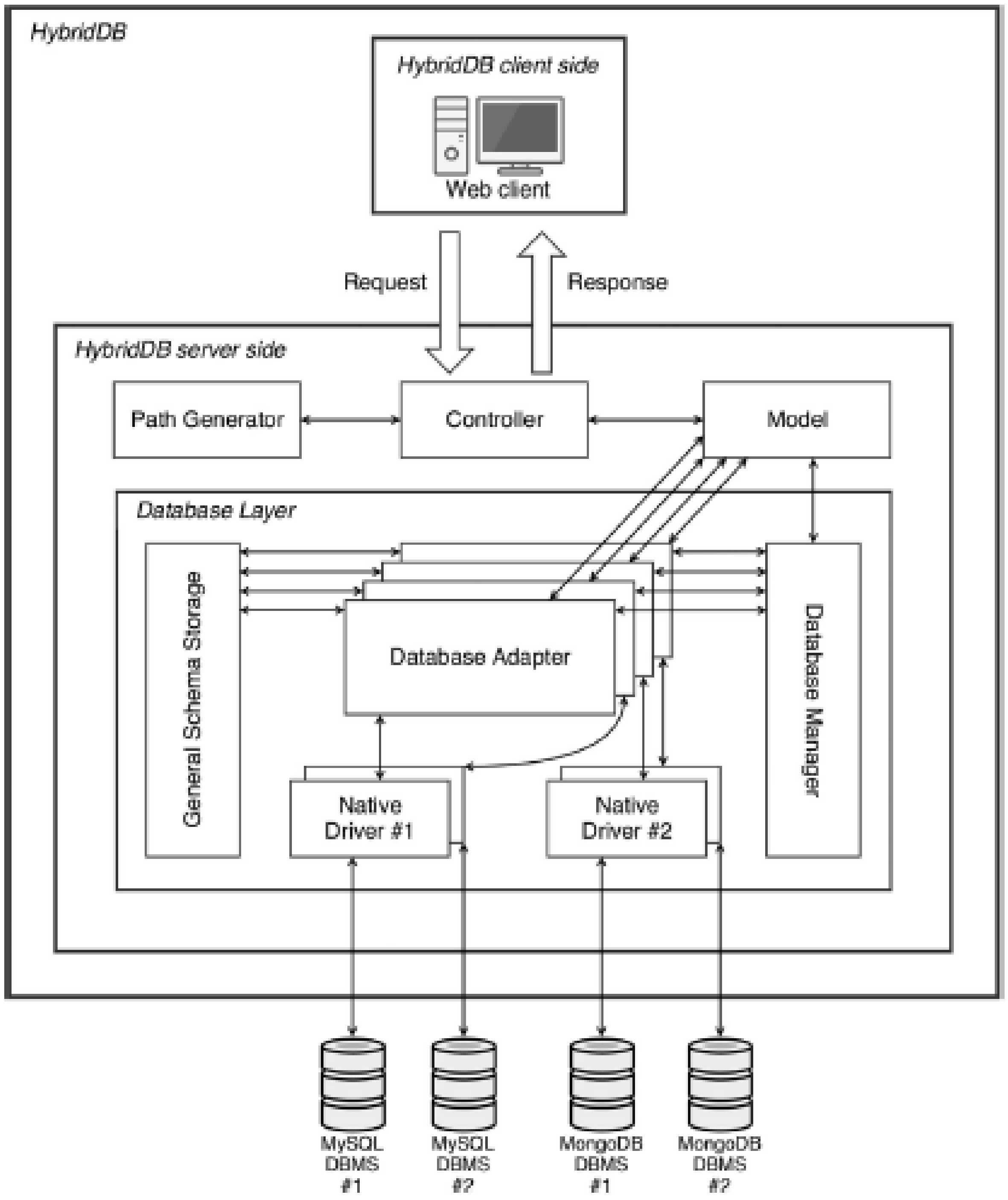


Figure 2.13 Architecture of HybridDB (Source: Vathy-Fogarassy and Hugyák, 2017).

The major advantage of this proposed model is that a user can readily describe the semantical association between the source database systems without the need to know the internal structure of the source databases before being competent to recover data from them.

The drawback of the research is that the recommended solution by this merger approach does not support joins and aggregates across data sources; it only collects data from different separated database management systems according to the filtering options and migrates them.

According to Vathy-Fogarassy & Hugyák (2017) in the existing edition of the HybridDB, data cached in MySQL and MongoDB databases can be integrated, but further research is required in terms of development of a more concise and robust model which requires integration of any other relational or NoSQL database management systems into the existing architecture.

2.8.11 An integration approach of hybrid databases based on SQL in cloud computing environment

Li and Gu (2019) worked on integrating multidatabases architecture which involved MySQL relational database, MongoDB and Redis NoSQL systems. The technique used by the researchers provided mechanisms for semantic transformation between SQL and MongoDB's Java Language API which helps in adding the SQL features on top these NoSQL databases and constructing the Redis NoSQL indexes to suit the transformation which leverages an effective reduced development complexity and also improves development efficiency of the software systems with multidatabases in cloud computing environment. The key algorithms were presented in the form of pseudo program code and the sytem was validated based on qualitative comparison with existing works and quantitative evaluation of the developed model.

The qualitative comparison involved the juxtaposition of the Unity architecture by Lawrence (2014) and the MSI architecture by the researchers; Li and Gu (2019). The features of the Unity architecture were compared with that of the MSI technique and the results obtained from the analysis of the two systems showed that MSI architecture has comparative advantage in terms of the incorporated data sources, the volume or size of data adopted in the research and query optimization. The quantitative evaluation of the performance of the MSI model involved; testing of the response time of the single data sources and that of the MSI model and testing the delay time using the APIs (Application Program Interface) of the prototype system. The results obtained by time analysis of accessing MongoDB and Redis showed that the individual systems were faster than MSI and Li and Gu (2019) suggested that data cache mechanism be improved which will help increase optimization and advancement to the field of database research.

As shown in Figure 2.14, thhe MSI architecture principally includes eight components; API dispatcher component, SQL parser component, SQL optimizer component, SQL router component, SQL executor component, DBMS (database management system) adapter component, result merger component, and meta-data management component.

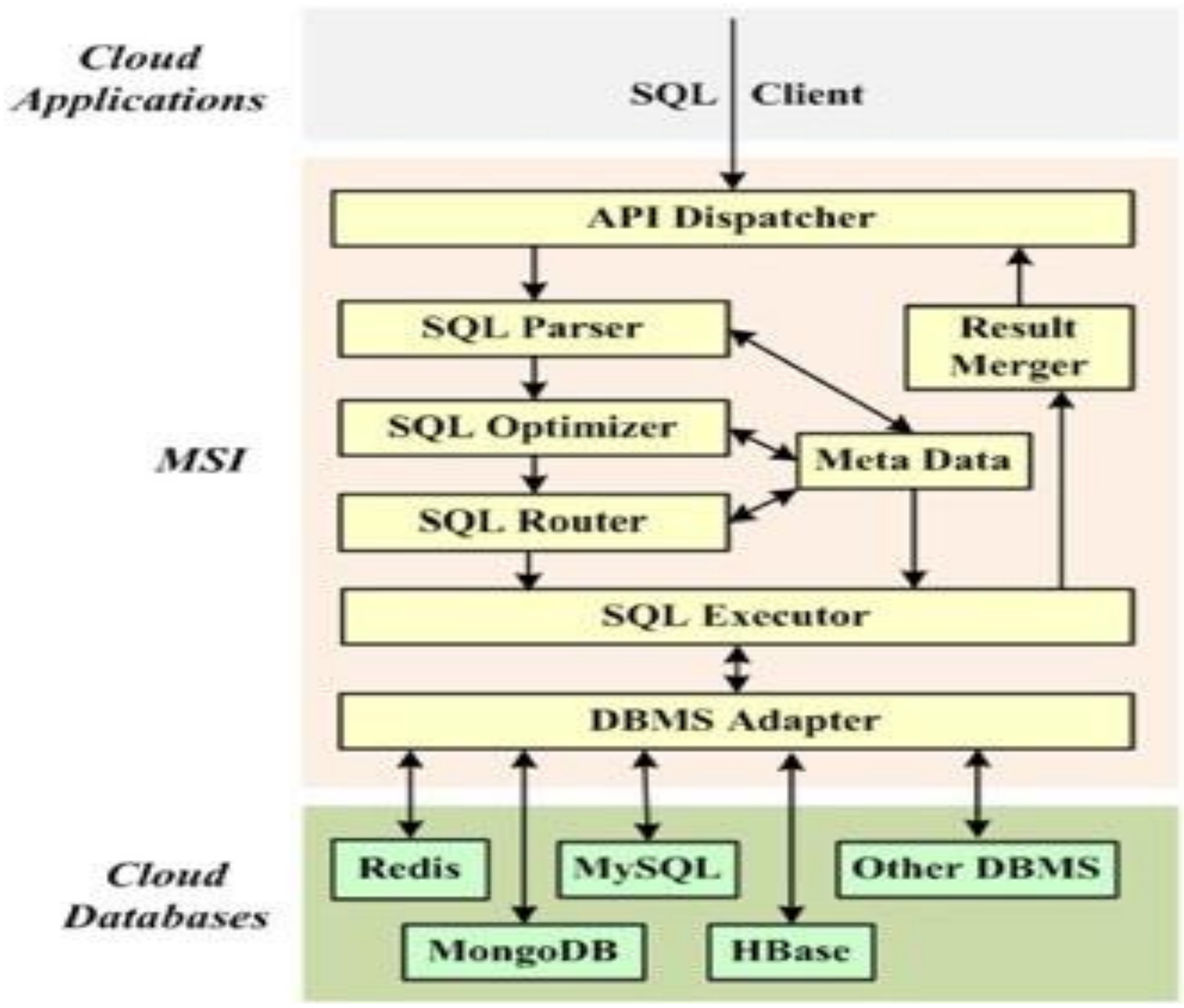


Figure 2.14 MSI (Multiple Sources Integration) Architecture (Source: Li and Gu, 2019).

According to Li and Gu (2019), “modern software development for services computing and cloud computing software systems is no longer based on a single database but on existing multidatabases and this convergence needs new software architecture and framework design”. The incorporation technique of SQL and NoSQL data-stores proposed by Li and Gu (2019) entitled MSI (Multiple Sources Integration), is based on the core layers of the operating system and the various hybrid database management systems. The SQL statements are the MSI's input and the result of the SQL process is the MSI's output.

Application patterns

As stated by Li and Gu (2019) the MSI can be applied in three main patterns and they include; OLTP (Online Transaction Processing), Big data analysis and Hybrid pattern.

OLTP pattern

As depicted in Figure 2.15, MSI is chiefly used as a support tool for OLTP (on-line transaction processing) application. For instance, its application is relevant in development of applications for e-commerce, e-government, internet of things etc. In these scenarios, several database systems are frequently drawn in, including relational databases and NoSQL database management systems.

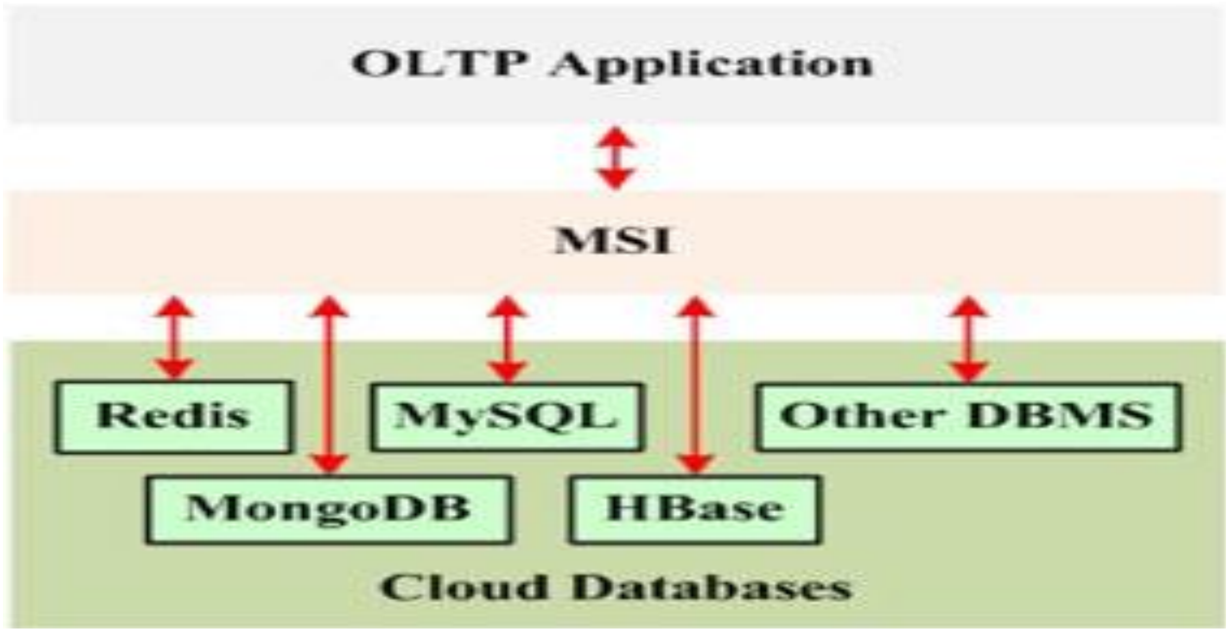


Figure 2.15 OLTP Application (Source: Li and Gu, 2019).

Big data analysis pattern

As shown in Figure 2.16, the MSI can be used as a support tool for big data analysis application, accessing data warehouse system, such as HBase NoSQL database, by SQL. Interchangeably, it can be applied in data synchronizer, which transfers data from the OLTP database system to data warehouse system.

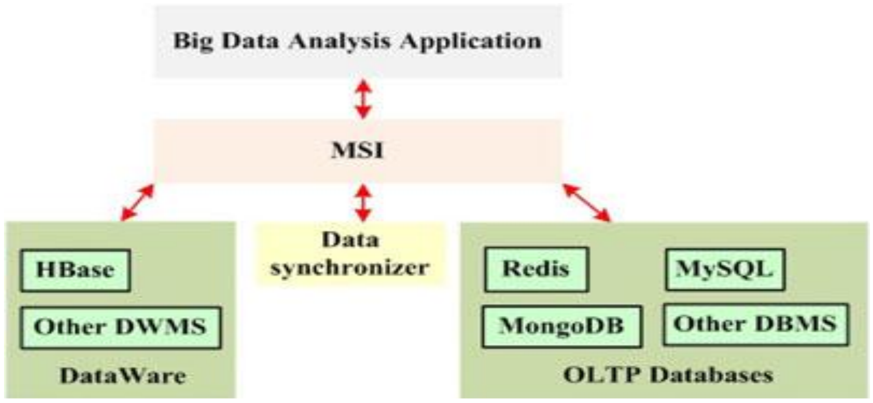


Figure 2.16 Big Data Analysis Application (Source: Li and Gu, 2019).

Hybrid pattern

Hybrid pattern is a co-existence way of the OLTP pattern and the big data analysis pattern, which is also the most familiar style in handy applications. Alternatively, as highlighted in Figure 2.17, the MSI functions as a link connecting application layer involving OLTP application and big data analysis application) and data storage layer (including OLTP database systems and data warehouse system).

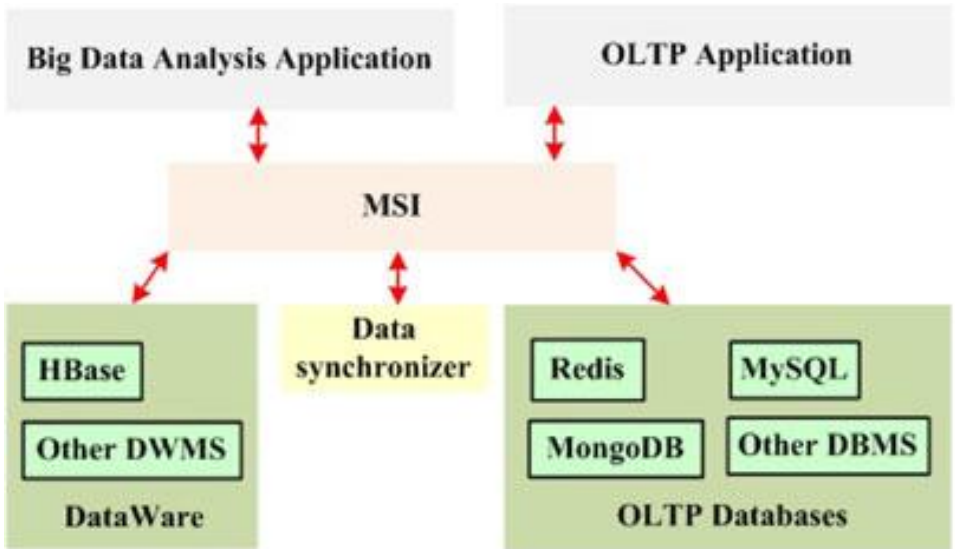


Figure 2.17 Hybrid Pattern (Source: Li and Gu, 2019).

The MSI model proposed by Li and Gu (2019) provides access to multiple data sources simultaneously and has access to MySQL relational database, MongoDB and Redis NoSQL systems and also runs on a considerably large amount of data. For the purpose of further research, the authors recommended the addition of more NoSQL systems into their prototype system as the MongoDB and Redis sytems only represent the key-value and document NoSQL systems. The research did not include the graph-oriented and column-share NoSQL systems. Redis: a key-value pair NoSQL system is good for insertions but has low disk space as it stores to physical memory and is not effective for large transactions. It does not perform read-write operations as quickly as document and column-oriented systems. As pointed out by Li and Gu (2019) the MSI model has the downside of of reduced speed or delay in response time which is caused by parsing, optimization and routing of SQL. According to Li and Gu (2019) “ it is not valuable but alsofeasible to add the SQL feature to these NoSQL database systems at the expense of some performance so as to improve the development efficiency of software systems”.

Table 2.9 shows summary of the different works which were investigated by this researcher which helped in providing better insight on how to carry out this research.

Table 2.9 Summary of Related works

S/N	Year	Authors	Technique	Results	Limitations
1	2011	Zhang *et al*.	Integration by use of data dictionary of column-oriented NoSQL system (Cassandra) and relational model (oracle database)	Affixing middleware which affords a unified interface	Poor scalability leading to slow system Cassandra + Oracle
2	2013	Liu and Vitolo	Graph data warehouse by use of ‘Graph Cube’ which incorporates graphs with tables	GDML & GDDL in order to deliver a local copy of data and to provide a graph or graph segment.	incorporation of graphs as part of fact tables without leveraging the core aspect of SQL which is join operation Graph + SQL
3	2013	Doshi *et al*.	Use of Instant Data Integrator (SQL to NewSQL sync) and Hibernate OGM (Object Grid Mapping) H-Base Driver	Blending the two types of data processing (scale-up and scale-out) using HBase and Hive	Slow system; has the problem of handling data at good speed HBase + Hive

S/N	Year	Authors	Technique	Results	Limitations
4	2014	Lawrence	Integration of SQL and NoSQL system ((MySQL + MongoDB) by use of virtualization technology	Unity architecture provides an interface for converting SQL queries to NoSQL APIs	Overhead in SQL transaction process which leads to reduced performance MongoDB + MySQL
5	2014	Zhao *et al.*	Schema conversion by means of table nesting and using denormalization method	Migration of data from SQL to NoSQL (MySQL to MongoDB)	overhead cost in terms of time spent during data migration from relational to any Big data management system MySQL + MongoDB
6	2016	Goyal *et al.*	Cross-platform relational model to NoSQL model using bloom filter mapping metadata from RDBMS to NoSQL DB.	RDBMS to NoSQL integration	Data loss as the model does not provide row and column comparisons over the entire data set. MySQL + MongoDB + Cassandra
7	2016	Stanescu *et al.*	Automatic mapping of MySQL tables to the equivalent NoSQL documents by means of middle layer using the INFORMATION_SCHEMA of MySQL (views)	Integration of MySQL and MongoDB by exploiting the 1:1 (one to one) 1:M (one to many) and N:M (many to many) relationship of RDBMS	Inability to experiment on complex databases using many tables with large number of databases. Lack of CRUD function MySQL + MongoDB
8	2013	Sanobar and Vanita	SQL support over NoSQL with the help of metadata by using an interface (Java application layer and NoSQL database MongoDB)	MySQL + MongoDB integration by embedding a java-based application within NoSQL database (MongoDB) with the help of middleware	Lack of trial of other NoSQL systems apart from MongoDB MySQL + MongoDB
9	2013	Vilaca *et al.*	DQE (Distributed Query Engine) for running SQL queries on top a NoSQL database while maintaining its scalability and schema flexibility	Integration of Apache Derby's query engine with full SQL support and HBase NoSQL database	The DQE has a slightly reduced throughput than HBase. DQE incurs greater overhead operationally due to manual optimization and denormalization caused by the Apache Derby's relational operator, filtering and secondary indexes Apache Derby+HBase

S/N	Year	Authors	Technique	Results	Limitations
10	2017	Vathy-Fogarassy & Hugyák	Implementation of a uniform data access interface for MySQL and MongoDB systems by applying the MVC (Model-View-Controller) using Node.js for the server-side framework and AngularJS for the client-side JavaScript framework.	HybridDB architectural design was modeled by incorporating JSON JavaScript Object Notation) principle in mapping different data models into a common one.	Inability to perform operations like aggregations and joins across data sources. The system does not perform CRUD operations but only migrates data. MongoDB + MySQL
11	2019	Li & Gu	Integration of multidatabases architecture involving MySQL, MongoDB and Redis systems by semantic transformation of MySQL and MongoDB's Java Language API and reconstruction of Redis indexes to comply with the transformation.	MSI architecture provides access to multiple data sources simultaneously	Reduced speed or delay in response time caused by parsing, optimization and routing of SQL. The Model requires additional NoSQL systems to increase performance and development efficiency. MySQL + MongoDB + Redis

2.9 Summary of Literature Review/ Research Gap

The investigation carried out shows that NoSQL systems are not replacements for the relational database systems as each could be used to perform specific purposes with optimum output. For some companies with need for a very consistent system which can hold well-structured data sets, the relational databases are still the best option for them. The NoSQL systems with their speed, availability and fail-over options serve the purpose of delivering run-time purposes of good throughput.The reviewed works did not perform interoperability among the different NoSQL systems and SQL system; this has been identified as a gap in knowledge as benefits accruing from exploration into a cohesive interoperable system has not been addressed.

CHAPTER THREE

METHODOLOGY AND SYSTEM ANALYSIS

3.1 Methodology Adopted

According to Bu *et al*., (2010), numerous data analysis techniques require iterative computations, including PageRank, HITS (Hypertext-Induced Topic Search), recursive relational queries, clustering, neural-network analysis, and social network analysis, and network traffic analysis. These techniques have a common trait: data are processed iteratively until the computation satisfies a convergence or stopping condition.

Relational algebra which is already in use in the SQL system is the basis for operations in the RDBMS. Statistical laws for example commutativity, distributivity and associativity make for operations in the NoSQL systems aided for interoperability of the systems. These mathematical/statistical concepts will be simplified to suit the SQL and NoSQL systems which could be used to create a middle-layer which in effect is the integration of the systems. The system will be based on combining some of the common rules like union, intersection and other common functions and indices which are similar or applicable to both systems.SQL systems run on relational algebra which is how the constraints are made to work both the referential and integrity constraints. This is based primarily on set theorem which identifies a relation between different variables and links them up. The join operations are made possible through this method.

As shown in Figure 3.1, this research developed a novel methodology from the existing Iterative and Incremental Development (IID) method which is termed ‘Holistic Iterative and Incremental Approach for cross-platform modeling’. However, the Iterative and Incremental Model alone does not entirely capture the architectural process involving drivers of the NoSQL systems evaluated and used in the implementation aspect of this study. Therefore, this research conceptualized additional processes that make the new methodology holistic and appropriate for the cross-platform development. Figure 3.1 captures the Iterative and Incremental approach as well as the researcher’s additions which is titled Holistic Iterative and Incremental Approach for Crosss-platform Modeling.

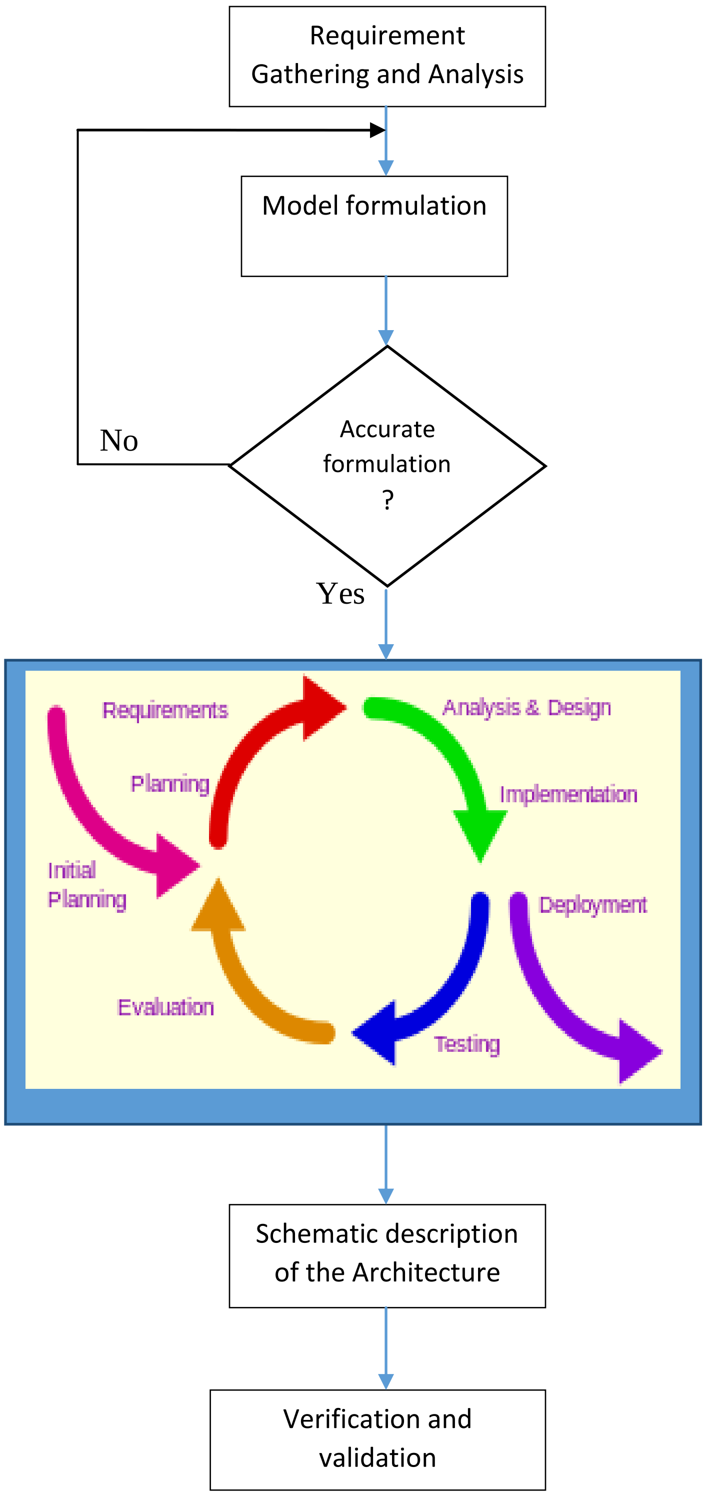


Figure 3.1 Holistic Iterative and Incremental Approach for Cross-platform Modeling

On the methodology for software development, this research adopted an Iterative and incremental approach. As the study demands accurate throughput, this approach offers a more reliable outcome as any error can be revisited and sorted while still carrying out other aspects of the work. The Iterative and Incremental model offers a continuous loop of development and testing; then the repeat of the process in order to allow for changes to be made to the system.

The research included comparison of different tools which were used to analyze for performance NoSQL systems. The relational algebra which is the interface to where data is stored in the SQL database was used as a tool in providing solutions to data sets for both the relational and non-relational data models using DDL (Data Definition Language) like insert, update and delete. Mathematical algorithms which combine set theorems for both SQL and NoSQL were used.

3.2 Analysis of Existing System

Sanobar & Vanita (2013) integrated MySQL RDBMS (Relational Database Management System) and MongoDB (Document store NoSQL) systems by means of application of a middleware layer across the two systems. The systems employed the use of middleware between the front-end (application layer) and the back-end system (database layer). The existing system leveraged on the gains of the SQL system which include; maintenance of data integrity and scalability and provision of unparalleled feature set and also the benefits of MongoDB whichserves majorly for read-intensive operations, and ability to store large amount of semi-structured and unstructured data. The MongoDB has an architectural design which supports BSON (Binary encoded JSON) data structures to store complex datatypes, supports powerful and complex query language and has high speed access to mass data, stores and distributes large binary files like images and videos. MongoDB supports indexing over embedded objects and arrays which provides a special feature for arrays called multikeys.

The Database Integration Architecture in Figure 3.2 implemented a package which serves as an interface between java application layer; which is the API (Application Prrogramming Interface) and NoSQL database (MongoDB).

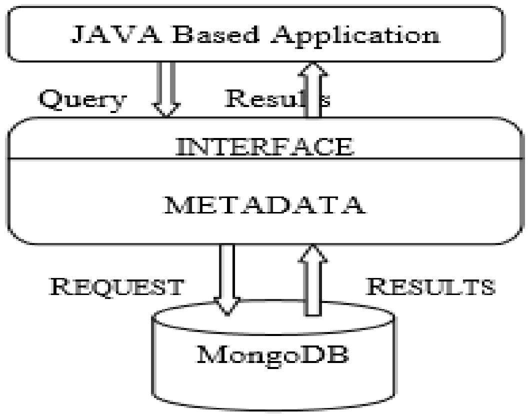
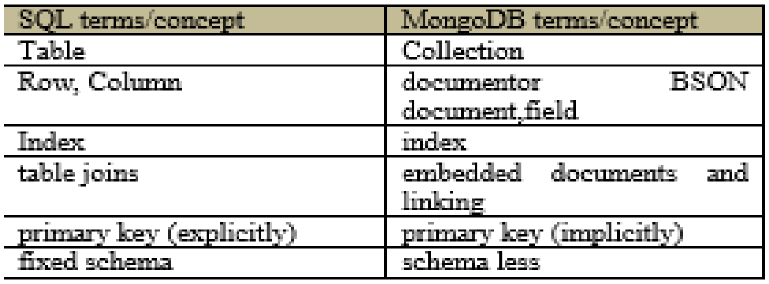


Figure 3.2 Database Integration Architecture (Source: Sanobar and Vanita, 2013).

Sanobar and Vanita (2013) did a comparison of MySQL and MongoDB systems based on three parameters; 1) Terms/Concept 2) Schema Statements and 3) performance.

Table 3.1 which is based on Terms/Concept of the systems explained the concept of the two systems as they relate to each other (based on their equivalence). The table in MySQL system is the equivalent of collection in MongoDB. Row, Column is the BSON document in MongoDB while indexes are on the same level for both systems.

Table 3.1: Terms/Concept of RDBMS and MongoDB Implementation (Source: Sanobar and Vanita, 2013).



SQL terms/concept	MongoDB terms/concept
Table	Collection
Row, Column	documentor BSON document,field
Index	index
table joins	embedded documents and linking
primary key (explicitly)	primary key (implicitly)
fixed schema	schema less

Table 3.2 outlined commands used for different operations by MySQL and MongoDB management systems. API (Application Programming Interface) MongoDB posseses its own query language named Mongo Query Language. In order to recover certain documents from a database collection, a query document is added containing the fields that the preferred documents should match. For example,

Insert Command

db.users.insert ({ user id:”abc123”, age: 55, status:”A”})

Drop Command

db.users.drop ()

Select Command

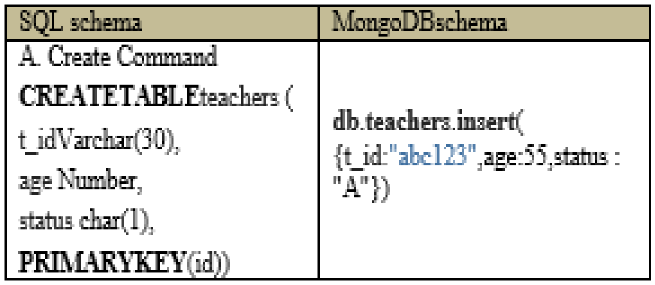
db.users.find ({ status:”A”, age: 55})

Delete Command

db.users.remove ({ status:”A”})

MongoDB uses a RESTful (RepresentationalState Transfer) API. RESTful is an architecture style for designing networked applications which relies on a stateless, client-server, cacheable communications protocol (e.g., the HTTPprotocol). RESTful applications use HTTP requests to post, read data and delete data.

Table 3.2: Example of Schema Statements for RDBMS and MongoDB (Source: Sanobar and Vanita, 2013).



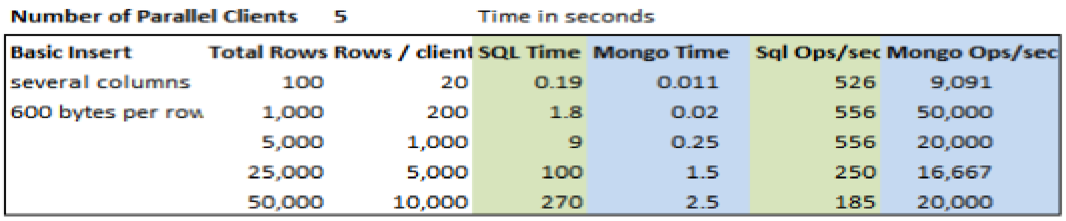
SQL schema	MongoDBschema
A. Create Command **CREATETABLE**teachers (t_idVarchar(30), age Number, status char(1), **PRIMARYKEY**(id))	**db.teachers.insert(** {t_id:"abc123",age:55,status : "A"})

B.DROP Command **DROP TABLE teachers**	**db.teachers.drop()**
C.INSERT Command **INSERTINTO**teachers(t_id, age,status) **VALUES** ("a123",45,"A")	**db.teachers.insert(** {t_id:"a123",age:45,status:"A" })

D.SELECT Command
SELECTt_id, status,age
FROMteachers
	db.teachers.find({ },{t_id:1,
	status:"B",age:45 })

E.DELETE Command **DELETEFROM** teachers **WHERE** status ="D"	**db.teachers.remove(** { status:"D" })

The system analysis was based on performance where testing based on insertion speed as shown in figure 3.3 were carried out. The tests used 100 to 50,000 records to ascertain speed of the two systems under comparison which are MySQL and MongoDB.



Number of Parallel Clients 5 Time in seconds

Basic Insert	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec
several columns	100	20	0.19	0.011	526	9,091
600 bytes per row	1,000	200	1.8	0.02	556	50,000
	5,000	1,000	9	0.25	556	20,000
	25,000	5,000	100	1.5	250	16,667
	50,000	10,000	270	2.5	185	20,000

Figure 3.3 Insertion Speed Comparisons for MySQL and MongoDB (Sanobar and Vanita, 2013).

Other performance metrics were used as shown in Figure 3.4 where completion of the different Insertion times for MySQL and MongoDB were tested. Also, Figure 3.5 outlined the query speed comparisons conducted for MySQL and NoSQL systems.

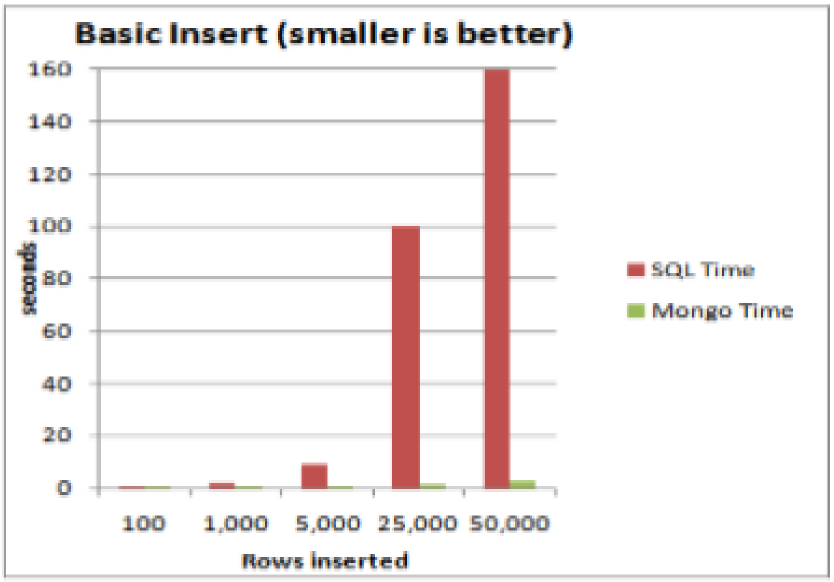
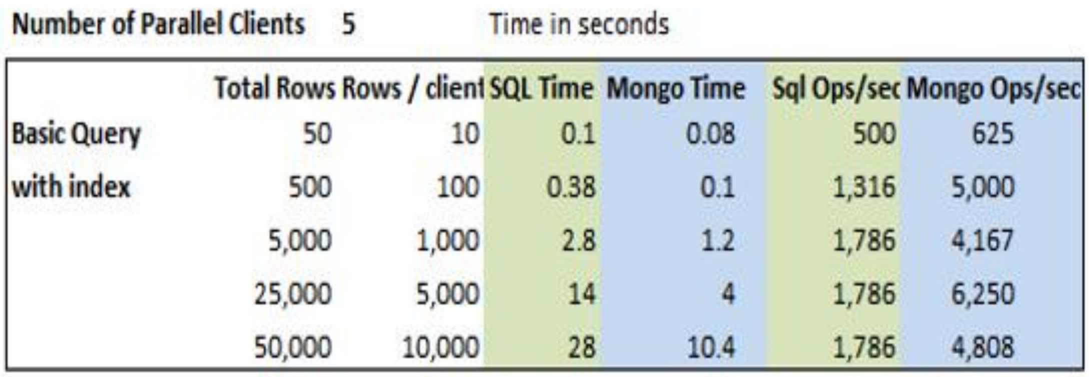


Figure 3.4 Insertion Time for MySQL and MongoDB (Sanobar and Vanita, 2013)

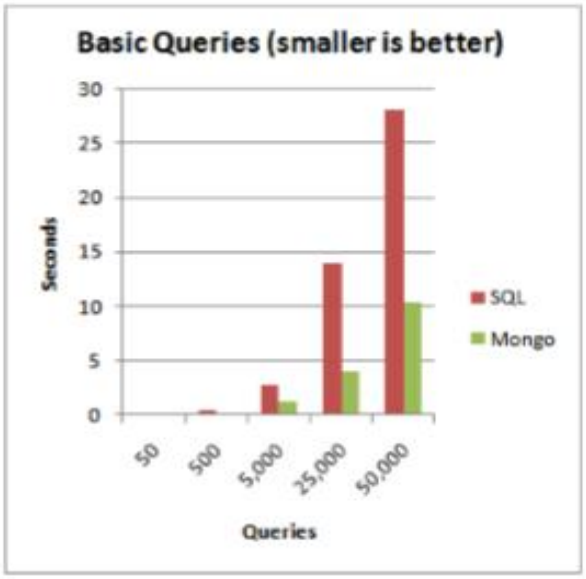


Number of Parallel Clients 5 Time in seconds

	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec
Basic Query	50	10	0.1	0.08	500	625
with index	500	100	0.38	0.1	1,316	5,000
	5,000	1,000	2.8	1.2	1,786	4,167
	25,000	5,000	14	4	1,786	6,250
	50,000	10,000	28	10.4	1,786	4,808

Figure 3.5 Query Speed Comparisons (Sanobar and Vanita, 2013)

The system is primarily designed for embedded java based application which requires database access. As shown in Figure 3.6, basic and complex query times were tested for MySQL and MongoDB. The database command fired from java based application is given as input to the interface acting as a middleware (Metadata). The interface parses the input and reformats the code in the format that is being required by the back-end database (MongoDB). The reformatted code consists of functions that directs back-end database to implement and maintain the database.



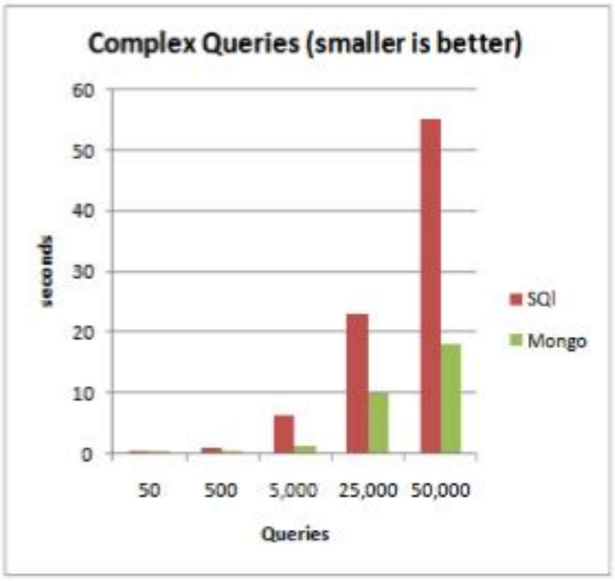


Figure 3.6 Basic and Complex Queries Time for MySQL and MongoDB (Sanobar and Vanita, 2013)

3.2.1 Advantages of the Existing System

1) The system offers an integrated architecture for SQL and NoSQL systems.

2) Embedded applications for example Java could be used for query parsing and interoperability between RDBMS model and NoSQL model.

3) The existing system showed the strength and weakness of MySQL and MongoDB and how the knowledge of this could be leveraged upon for improvement.

3.2.2 Disadvantages of the Existing System

1) The existing system proposed integration only for two systems (MySQL and MongoDB).

2) The existing system showed performance analysis for MySQL and MongoDB but did not test for the integrated system.

3) The basic query operations which comprise of; insert, read, update and delete were not conducted.

4) Actual test results which could have been used as benchmark for comparison with the new system were not included.

3.3 Analysis of the New System

The new system combined the various desirable attributes of the two systems; SQL and NoSQL systems using a novel approach which is obtained from the already existing iterative and incremental model called ‘Holistic Iterative and Incremental Approach for Cross-platform Modeling’.

Some systems have been developed to handle the growing amounts and complexity of data example is the Unity system by Lawrence (2014) and DQE (Distributed Query Engine) by Vilaca *et al*., (2013) and also the MSI model by Li and Gu (2019).

NoSQL^2; the new hybrid system developed by this research titled ‘Design and Implementation of a cross-platform Database system for SQL and NoSQL Interoperability’ encompasses the features of both relational and all the different types of NoSQL systems. The new hybrid system serves the purpose of handling transactions, consistency and availability. The system offers automatic load balancing which handles the velocity (speed), volume (amount) and variety (different structures of data). The system performs analytics and search and has been tested for Usability.

The new system leverages an enhanced hybrid storage mechanism which is flexible, reliable with easy manageability. The new system offers portability as it enables SQL which is the primary repository of data to be incorporated into the NoSQL database. This interoperability of the systems offer a distributed, fault-tolerant system with failover option made possible by the scalabilty of the system. As a result of extra nodes made available to the system whereby data can be spread across different systems optimized for the purpose of handling data according to strength, need and requirement of the dataset, the problem of bottlenexk is made redundant and performance is increased. Throughput which is the amount of data transferred within a given period of time is increased.

The new system having been built exclusively for enhanced data dissemination and storage with optimal low latency performs automatic aggregation which makes for responsiveness and quick delivery of data. The hrbrid system allows CRUD (Create, Read, Update and Delete) operations and sharding (horizontal partitioning) whereby different sets of values are stored in different tables according to use case or suitability and conformation to the storage engines to be carried out. Denormalization; a process that allows for increased response time for data recovery while sustaining good system throughput for row insertions, updates and deletions works for the new system. The new system provides good elasticity and is also reliable, consistent and maximally available as the platform is built for persistence.

3.3.1 Justification of the New System

The CRUD function allows for table creation and update and also allows for insertions to be deleted by the admin. This function is one of the major characterizes of the SQL system and will be fused into the hybrid system to be developed.

The optimization function allows the SQL system to automatically fast track query processes within the management system and the ACID property makes for compactness of the management system. All these attributes will allow for an efficient cross-platform system of SQL and NoSQL management systems. The BASE function of the NoSQL system leverages a system which is both available and secondarily consistent. The PACELC function favours a system which allows for partitioning of the database systems, Else a system that has low latency and at the same time consistent. The CAP (Consistency, Availability, Partition tolerance) function of the NoSQL system allows for two out of the three functionalities of the system, it favours either consistency and partition tolerance or availability and partition tolerance but never allows for consistency and availability at the same time. The new system will allow for consistency, availability and partition tolerance to function at the same time. Sharding which is horizontal partitioning will over rule the vertical partitioning of the NoSQL system and will offer both speed and space and will also be flexible while also being secure.

3.4 High Level Model of the New System

Figure 3.7 depicts the high level model of the proposed system. The model incorporates the various characteristics of the relational database management system with that of the NoSQL systems which makes the new high level model. The inherent characteristics of the SQL system are; ACID (Atomicity, Consistency, Isolation and Durability), CRUD (Create, Read, Update and Delete) and also Optimization wherby the query optimizer tries to determine the most efficient way to execute a given query considering the possible query plans. The NoSQL systems have their own unique and identifiable properties which are BASE (Basically Available, Soft State and Eventual Consistency), SHARD (horizontal partitioning of the database) where different sets of values are stored in different NoSQL tables, PACELC (Partition, Availability, Consistency, Else; Latency and Consistency), MAPREDUCE (aggregation) and CAP (Consistency, Availability and Partition Tolerance).

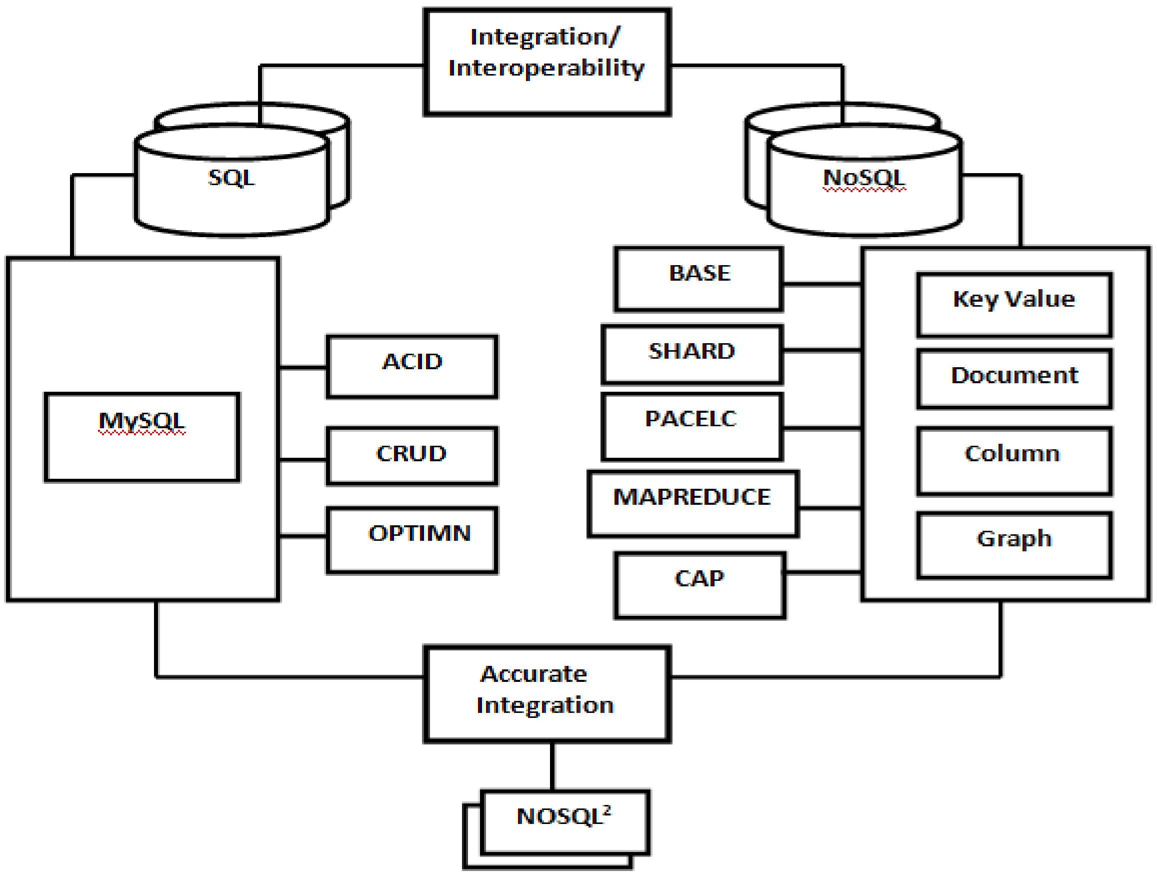


Figure 3.7 High Level Model of the Proposed System

3.5 Objectives of the New Model Design

The objectives of the new model design are as follows:

i) To provide an improved system capable of translating information across the various databases using a common syntax.

ii) The new model among other things will offer an integrated system which can reliably parse data/information (interoperable system) within the system

iii) To discover the drivers and similarity of the various NoSQL systems which have been brought together and how well they can share/co-exist with the RDBMS in a user-friendly environment.

iv) An upgrade from the normal/known method of data storage and retrieval and also implementation of security within the developed system which is not obtainable in the different individual systems which have been combined/ integrated to arrive at the new model design.

3.6 Decomposition and Cohesion of the High Level Model

The high level model of Figure 3.7 contains a holistic overview of the conceptualization of the new high level model which is being designed and implemented. The several types/features of the NoSQL systems were duly highlighted and they are: Key Value, Document, Column and Graph NoSQL database systems. Their key characteristics/attributes are BASE (Basically Available, Soft State, and Eventual consistency), SHARD (Horizontal Partitioning of the database), PACELC (Partition, Availability, and Consistency, Else Latency and Consistency), Mapreduce (aggregation using algorithm suitable to type of database involved) and CAP (Consistency, Availability and Partition tolerance). These features are the decomposition of the various NoSQL systems involved which when synchronized, gives a proper representation of the High Level Model concerning the NoSQL part of the implementation process.

The SQL/RDBMS (Relational Database Management System) has features like; ACID (atomicity, consistency, isolation and durability), also transactions (the basic unit of work) of a database are the major considerations of the RDBMS as a process is considered to have been fully committed when the transaction is completed. Also, CRUD (create, read, update and delete) are major querying formulations of the SQL (structured query language) management system.

This is because this procedure allows for insertions/write to the database and allows for deletion to be made to the RDBMS. The CRUD makes it possible for the database to be modeled and also periodic inputs known as updates are made to the management system. OPTMIN (optimization) allows for the database to manage itself efficiently. Optimization reduces overhead which could accrue as a result of querying the database. Join operations which could get complicated at a certain point in the database as a result of the structural dependencies or normalization are reduced as a result of the optimization process. OPTMIN offers a form of aggregation as it exhibits the ACID property of the database which is making the management system a wholesome system which performs optimally when the tables and the variables have been fully declared and defined.

These various attributes outlined were made to function together when the systems have been looked into. A proper cohesion of these distinctiveness leverages a unified system. The algorithm shows that the uniqueness of these systems running individually as a functional system offered when put together, a viable system capable of combating the weaknesses of all the individual database management systems.

3.7 Specifications

3.7.1 Database Development Tools

The High Level Model followed a Polyglot Persistence flow of software development. This is the combination of various applications coming together and serving different purposes with the sole aim of developing a functional and integrated application for data management. Various implementation models were considered but were not used for implementation as a result of compatibility issue. The tools that have been chosen are: PHP, XAMPP, Apache, Cassandra, MongoDB, Redis, MYSQL and Neo4j.

3.7.2 Database Systems and Structure

Figure 3.8 outlines the various database systems (back-end) and the API (Application Programming Interface) which is the PHP (front-end). The various systems were made interoperable with the help of the PHP scripting language which leverages communication amongst the different database engines. The Neo4j system is a graph-oriented NoSQL DBMS employed for special purpose of manaing graphical associations like multimedia and also social media. Graph-oriented databases are built to manage associations or relationships not to query data (Zafar *et al*., 2016).

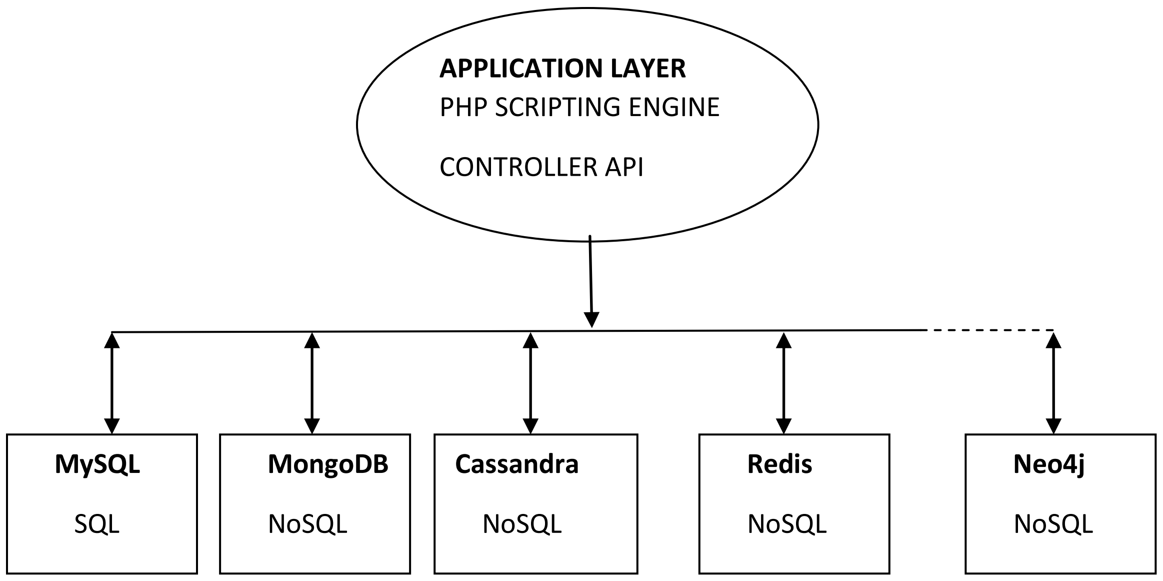


Figure 3.8 Database Systems and Structure

As shown in figure 3.9, the graph database management system are special purpose databases which characteristically handles complex relationship. Jayathilake *et al*., (2012) defined graph database as an arrangement of nodes, edges and properties to represent and hold data. This feature enables graph database to hold complex relationships which are difficult to absorb in other databases like the many-to-many relationships. According to Srivastava and Shekokar (2016) graph stores work well only with relationships requirements such as processing on social network.

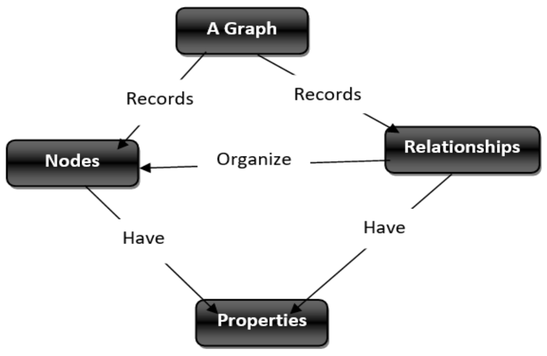


Figure 3.9 Layout of Graph Database (Source: Ejiofor and Okeke, 2016).

3.7.3 Mathematical Specifications

Mathematics is one of the most important differences among SQL, NoSQL, and NewSQL databases. On the other hand, one should note that, "too much mathematical rigor burdens a database implementation with unnecessary mathematics" (Kepner, *et al*., 2016). SQL is based upon basic binary operations such as union and intersection of special sets known as relations, which brings about the concept relational algebra.

A relation $R: A \longrightarrow B$ from a set $A = \{a_1, a_2, \dots, a_m\}$ to a set $B = \{b_1, b_2, \dots, b_n\}$ is a set of ordered pairs $R = \{(a_i, b_j): a_i$ is related to b_i and $a \in A, b \in B\}$. Operations on relations obey the usual laws of set operations. These laws include the commutative laws which states that

$R_1 \cup R_2 = R_2 \cup R_1$ and

$R_1 \cap R_2 = R_2 \cap R_1$, whenever R_1 and R_2 are given relations on a set A;

associative laws which states that

$(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$ and

$(R_1 \cap R_2) \cap R_3 = R_1 \cap (R_2 \cap R_3)$ whenever R_1, R_2, and R_3 are given relations on a set A; and

distributive laws which states that intersection is distributive over union, that is,

$R_1 \cap (R_2 \cup R_3) = (R_1 \cap R_2) \cup (R_1 \cap R_3)$ and union is distributive over intersection, thus

$R_1 \cup (R_2 \cap R_3) = (R_1 \cup R_2) \cap (R_1 \cup R_3)$ whenever R_1, R_2, and R_3 are given relations in A.

To represent a relation in A (that is from A to A) in a matrix or a graph which is the basis upon which the NoSQL operates, the matrix M_R was defined in the following way thus:

$$M_R = \begin{cases} m_{ij} = 1 & (a_i, b_j) \in R \\ m_{ij} = 0 & (a_i, b_j) \notin R \end{cases} \qquad 3.1$$

This usually gives a sparse matrix with ones as the nonzero entries. Sparse matrices optimize storage and computing time in data processing.

Given sparse matrices A and B as in (1) that corresponds to the relation R_1 and R_2 respectively, we define matrix addition as

$$A \oplus B = \begin{cases} 1 & 1 \in A \; or \; 1 \in B \\ 0 & 0 \in A \; and \; 0 \in B \end{cases} \qquad 3.2$$

It says thus, the result is 1 whenever there is 1 in any of the matrices and 0 whenever both of the matrices have 0 in them. The matrix obtained will be equal to the matrix gotten when $R_1 \cup R_2$ is represented in matrix form.

Also, the element-wise multiplication of the sparse matrices A and B was defined:

$$A \otimes B = \begin{cases} 1 & 1 \in A \; and \; 1 \in B \\ 0 & 0 \in A \; or \; 0 \in B \end{cases} \qquad 3.3$$

The result is 1 whenever 1 is in both matrices and 0 if 0 is in either of the matrix. The matrix obtained in this operation is equivalent to the matrix obtained when $R_1 \cap R_2$ is represented in matrix form.

Clearly, these operations satisfy the commutative law, associative law and distributive law since given the sparse matrices A, B and C as defined in (3.1):

i. $A \oplus B = \begin{cases} 1 & 1 \in A \; or \; 1 \in B \\ 0 & 0 \in A \; and \; 0 \in B \end{cases}$

$$= \begin{cases} 1 & 1 \in B \; or \; 1 \in A \\ 0 & 0 \in B \; and \; 0 \in A \end{cases} = B \oplus A$$

and

$$A \otimes B = \begin{cases} 1 & 1 \in A \; and \; 1 \in B \\ 0 & 0 \in A \; or \; 0 \in B \end{cases} = \begin{cases} 1 & 1 \in B \; and \; 1 \in A \\ 0 & 0 \in B \; or \; 0 \in A \end{cases} = B \otimes A$$

Indicating that the operations $\oplus$ and $\otimes$ are commutative.

ii. $(A \oplus B) \oplus C = \left(\begin{cases} 1 & 1 \in A \; or \; 1 \in B \\ 0 & 0 \in A \; and \; 0 \in B \end{cases}\right) \oplus C$

$$= \begin{cases} 1 & 1 \in A \; or \; 1 \in B \; or \; C \\ 0 & 0 \in B \; and \; 0 \in A \; or \; C \end{cases}$$

$$= A \oplus \left(\begin{cases} 1 & 1 \in B \; or \; 1 \in C \\ 0 & 0 \in B \; and \; 0 \in C \end{cases}\right)$$

$$= A \oplus (B \oplus C)$$

and

iii. $(A \otimes B) \otimes C = \left(\begin{cases} 1 & 1 \in A \; and \; 1 \in B \\ 0 & 0 \in A \; or \; 0 \in B \end{cases}\right) \otimes C$

$$= \begin{cases} 1 & 1 \in A \; and \; 1 \in B \; and \; C \\ 0 & 0 \in B \; or \; 0 \in A \; or \; C \end{cases}$$

$$= A \otimes \left(\begin{cases} 1 & 1 \in B \; and \; 1 \in C \\ 0 & 0 \in B \; or \; 0 \in C \end{cases}\right)$$

$$= A \otimes (B \otimes C)$$

Indicating that the sparse matrices is associative under the operations $\oplus$ and $\otimes$.

Addition is distributive over element-wise multiplication

$(A \oplus (B \; \otimes \; C) =$

$$= A \oplus \left(\begin{cases} 1 & 1 \in B \; and \; 1 \in C \\ 0 & 0 \in B \; or \; 0 \in C \end{cases}\right)$$

$$\left(\begin{cases} 1 \; if & 1 \in A \; or \; 1 \in B \; and \; 1 \in C \\ 0 \; if & 0 \in A \; and \; 0 \in B \; or \; 0 \; \in C \end{cases}\right)$$

$$= \left(\begin{cases} 1 \; if & 1 \in A \; or \; 1 \in B \; and \; 1 \in A \; or \; 1 \in C \\ 0 \; if & 0 \in A \; or \; 0 \in B \; and \; 0 \; \in A \; or \; 0 \; \in C \end{cases}\right)$$

$$= \begin{cases} 1 \; if & 1 \in A \; or \; 1 \in B \\ 0 \; if & 0 \in A \; or \; 0 \in B \end{cases} \quad \otimes \quad \left(\begin{cases} 1 & if \; 1 \in \; A \; or \; 1 \in C \\ 0 & if \; \; 0 \in A \; or \; 0 \in C \end{cases}\right)$$

$$= (A \oplus B) \otimes (A \oplus C)$$

Element-wise multiplication is distributive over addition

$$A \otimes (B \oplus C)$$

$$A \otimes \left(\begin{cases} 1 & if \quad 1 \in B \; or \; 1 \in C \\ 0 & if \quad 0 \in B \; and \; 0 \in C \end{cases}\right)$$

$$= \left(\begin{cases} 1 \; if & 1 \in A \; and \; 1 \in B \; or \; 1 \in C \\ 0 \; if & 0 \in A \; or \; 0 \in B \; or \; 0 \in C \end{cases}\right)$$

$$= \begin{cases} 1 \; if & 1 \in A \; and \; 1 \in B \\ 0 \; if & 0 \in A \; or \; 0 \in B \end{cases} \quad \oplus \quad \left(\begin{cases} 1 & if \; 1 \in A \; and \; 1 \in C \\ 0 & if \; 0 \in A \; or \; 0 \in C \end{cases}\right)$$

$$= (A \otimes B) \oplus (A \oplus C)$$

This set of sparse matrices has its identity with respect to $\oplus$, denoted as $\mathbb{0}$, where $\mathbb{0}$ is defined as a matrix of all 0 and its identity with respect to $\otimes$, denoted as $\mathbb{1}$, where $\mathbb{1}$ is defined as a matrix of all 1 since for any sparse matrix A as defined in (1):

$$A \oplus \mathbb{0} = A = \mathbb{0} \oplus A$$

and

$$A \otimes \mathbb{1} = A = \mathbb{1} \otimes A$$

Furthermore, these sparse matrices have a multiplicative annihilator $\mathbb{0}$ such that

$$A \otimes \mathbb{0} = \mathbb{0} = \mathbb{0} \otimes A$$

Table 3.3 shows the focus areas of SQL, NoSQL/NewSQL, some existing systems reviewed in this study and the new hybrid model developed by this research which included data model, usability tests and the type of applications the various systems are able to handle.

Table 3.3 Focus areas of SQL, NoSQL, Existing Systems and New Hybrid System

	SQL	NoSQL	Existing Systems	New Hybrid System
Example	MySQL	MongoDB	Unity System	NoSQL^2
Application	Transactions	Search	Analysis	All
Data model	Relational tables	Key-value pairs/Graph	Sparse Matrices	Associative Arrays
Consistency	Yes	No	No	Yes
Volume	No	Yes	Yes	Yes
Velocity	No	Yes	Yes	Yes
Variety	No	Yes	No	Yes
Analytics	No	No	Yes	Yes
Usability	Yes	No	No	Yes

3.7.4 Input/Output Format

Data is received and sent to the back-end system through the input forms on the screen provided by the Application Interface. Input is collected through the following forms updates by clicks;

Add New Records modules has sub modules;

Full Name

Gender

Phone Number

Email Address

These information are then forwarded to the back-end by clicking on;

Save Changes button or discarding the form by clicking on Close button

The form submits the Add New Records information and continues to other modules which are;

Add New store/Branch

Add New Sale

Add New Product

The system outputs information by clicking on View modules for example,;

View Purchase (displays purchase records and other information)

View Sales

3.7.5 Algorithm

Start the program

Run PHP Application

Perform SQL/NoSQL Commands

PHP Parses/ translates command

If error, return to PHP Application and correct then continue the process

If no error, proceed to the load balancer/ optimizer engine

Run file

Display result

End

Start Xampp application server for Apache and MySQL

Start MongoDB server

Start Cassandra server

Start Redis server

Start Neo4j server

End

3.8 Flowchart

Figure 3.10 is the program flowchart; a set of accepted standard graphics symbols representing the order of coded guidelines provided for the computer, facilitating it to conduct stipulated logical and arithmetic processes.

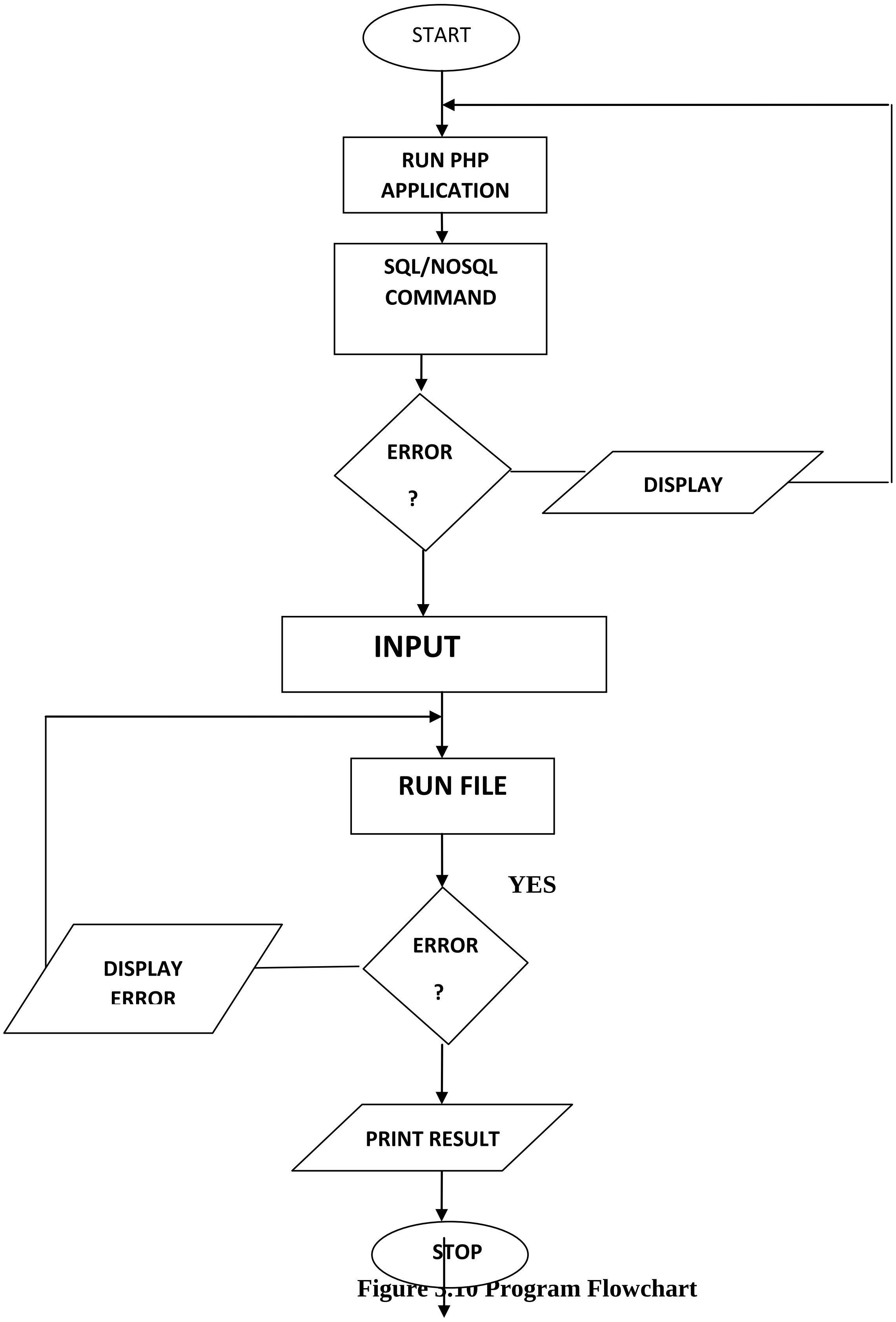


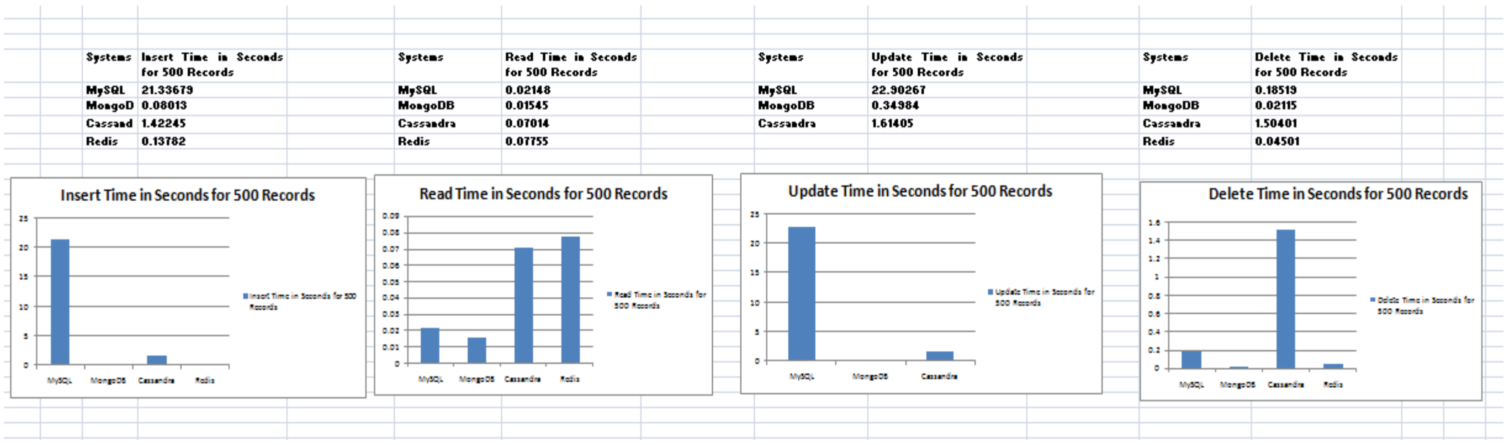
Figure 3.10 Program Flowchart

CHAPTER FOUR

SYSTEM DESIGN AND IMPLEMENTATION

4.1 Results

The test results in Figure 4.1 show CRUD operations performed using 500, 5,000 and 10,000 records for each database system involved (MySQL, MongoDB, Cassandra and Redis.



Systems	Insert Time in Seconds for 500 Records
MySQL	21.33679
MongoD	0.08013
Cassand	1.42245
Redis	0.13782

Systems	Read Time in Seconds for 500 Records
MySQL	0.02148
MongoDB	0.01545
Cassandra	0.07014
Redis	0.07755

Systems	Update Time in Seconds for 500 Records
MySQL	22.90267
MongoDB	0.34984
Cassandra	1.61405

Systems	Delete Time in Seconds for 500 Records
MySQL	0.18519
MongoDB	0.02115
Cassandra	1.50401
Redis	0.04501

Insert Time in Seconds for 500 Records
Read Time in Seconds for 500 Records
Update Time in Seconds for 500 Records
Delete Time in Seconds for 500 Records
MySQL
MongoDB
Cassandra
Redis

Figure 4.1 Query completion times for 500 records

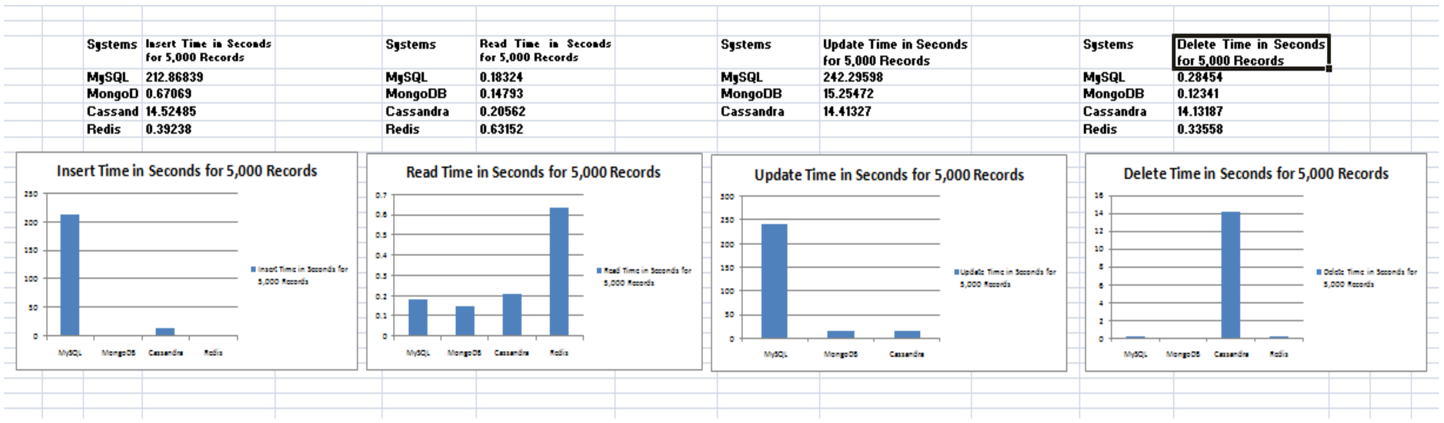
The test results show CRUD operations performed using 500 records for each database system involved (MySQL, MongoDB, Cassandra and Redis.

The insert query operation in Figure 4.1 showed that MySQL system completed insertion in 21.33679 seconds which makes it the slowest system amongst the four systems tested. MongoDB completed the operation in 0.08013 seconds which makes it the fastest, followed by Redis and Cassandra which completed their operations in 0.13782 and 1.42245 seconds respectively. MongoDB and Redis are good choices for insert operations.

The read query in Figure 4.1 shows that MongoDB performed fastest with completion time of 0.01545 seconds followed by MySQL 0.02148 seconds and Cassandra 0.07014 and Redis 0.07755 seconds. MySQL and MongoDB are good choices for read query operations.

The update query in Figure 4.1 shows MongoDB to be the fastest with finish time of 0.34984 seconds, followed by Cassandra 1.61405 and MySQL 22.90267 seconds. The completion times for MongoDB and Cassandra makes them a good choice for performing updates.

Delete query in Figure 4.1 has Cassandra as the slowest with 1.5040, followed by MySQL 0.18519 and Redis 0.04501. MongoDB performed fastest with 0.02115 seconds completion time.



Systems	Insert Time in Seconds for 5,000 Records
MySQL	212.86839
MongoD	0.67069
Cassand	14.52485
Redis	0.39238

Systems	Read Time in Seconds for 5,000 Records
MySQL	0.18324
MongoDB	0.14793
Cassandra	0.20562
Redis	0.63152

Systems	Update Time in Seconds for 5,000 Records
MySQL	242.29598
MongoDB	15.25472
Cassandra	14.41327

Systems	Delete Time in Seconds for 5,000 Records
MySQL	0.28454
MongoDB	0.12341
Cassandra	14.13187
Redis	0.33558

Insert Time in Seconds for 5,000 Records
Read Time in Seconds for 5,000 Records
Update Time in Seconds for 5,000 Records
Delete Time in Seconds for 5,000 Records
MySQL
MongoDB
Cassandra
Redis

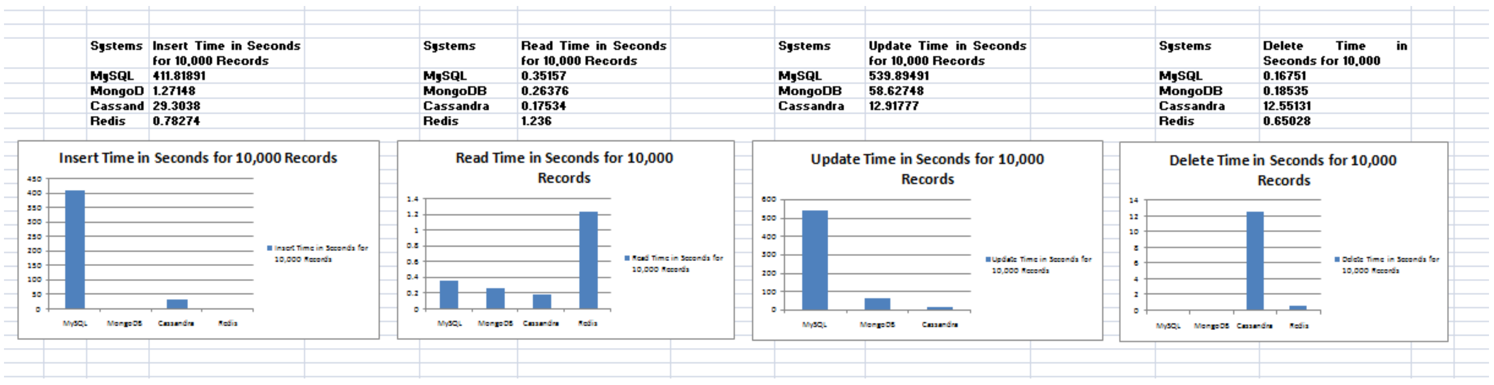
Figure 4.2 Query completion times for 5,000 records

The insert query operation in Figure 4.2 showed that MySQL system completed insertion in 212.86839 seconds which makes it the slowest system amongst the four systems tested. Redis completed the insert operation in 0.03928 seconds which makes it the fastest, followed by MongoDB and Cassandra which completed their operations in 0.67069 and 14.52485 seconds respectively. Redis and MongoDB are good choices for insert operations.

The read query in Figure 4.2 shows that MongoDB performed fastest with completion time of 0.14793 seconds followed by MySQL 0.18324 seconds and Cassandra 0.20562 and Redis 0.63152 seconds. MySQL and MongoDB are good choices for read query operations.

The update query in Figure 4.2 shows Cassandra to be the fastest with finish time of 14.41327 seconds, followed by MongoDB 15.25472 seconds and MySQL performed slowest with completion time of 242.29598 seconds. The completion times for MongoDB and Cassandra makes them a good choice for performing updates. Redis does not have an update feature; add and update feature are the same.

Delete query in Figure 4.2 has MongoDB as the fastest with completion time of 0.12341, followed by MySQL 0.28454 and Cassandra 0.33558 and Cassandra which completed in 14.13187 seconds.



Systems	Insert Time in Seconds for 10,000 Records
MySQL	411.81891
MongoD	1.27148
Cassand	29.3038
Redis	0.78274

Systems	Read Time in Seconds for 10,000 Records
MySQL	0.35157
MongoDB	0.26376
Cassandra	0.17534
Redis	1.236

Systems	Update Time in Seconds for 10,000 Records
MySQL	539.89491
MongoDB	58.62748
Cassandra	12.91777

Systems	Delete Time in Seconds for 10,000
MySQL	0.16751
MongoDB	0.18535
Cassandra	12.55131
Redis	0.65028

Insert Time in Seconds for 10,000 Records
Read Time in Seconds for 10,000 Records
Update Time in Seconds for 10,000 Records
Delete Time in Seconds for 10,000 Records
MySQL
MongoDB
Cassandra
Redis

Figure 4.3 Query completion times for 10,000 records

The insert query operation in Figure 4.3 showed that MySQL system completed insertion in 411.81891 seconds which makes it the slowest system amongst the four systems tested. Redis completed the insert operation in 0.78274 seconds which makes it the fastest, followed by MongoDB and Cassandra which completed their operations in 1.27148 and 29.3038 seconds respectively. Redis and MongoDB are good choices for insert operations.

The read query in Figure 4.3 shows that Cassandra performed fastest with completion time of 0.17534 seconds followed by MongoDB 0.26376 and MySQL 0.35157. Redis performed slowest with 1.236 completion time.

The update query in Figure 4.3 shows Cassandra to be the fastest with finish time of 12.91777 seconds, followed by MongoDB 58.62748 seconds and MySQL performed slowest with 539.89491. Redis does not have an update feature but was not tested for update.

Delete query in Figure 4.3 has MySQL as the fastest with completion time of 0.16751, followed by MongoDB with 0.18535 and Redis 0.65028. The slowest was Cassandra which completed in 12.55131 seconds.

4.2 Discussion

The findings of the research outlined performance comparisons conducted on the SQL and NoSQL systems. For insert operations, the test showed that Redis is faster for all the other systems followed by MongoDB. Redis System which is a key-value pair NoSQL system does not have update function in its build up and so performs update and insert uniformly. For this reason, the insertion operation is optimized for fast insertions. The Redis system only performs Add (create/insert), read and delete operations.For read query operations, MoongoDB and

MySQL performed faster than the other systems while Cassandra which was noted in the review chapter as being write-intensive outperformed all the other systems. The purpose of these tests was to ascertain the strengths and weaknesses of these systems to help proffer a distributed and interoperable database. The emphasis is not entirely on speed but on achieving a load balancing and fault-tolerant system based on the individual performances of the systems involved. Neo4j was tested alongside the other systems but showed heavy overhead in terms of completion time for very few records which indicates that Neo4j is not a good fit for query purposes. The Neo4j system does not have a functional update and delete operation at the application interface so deletion of records (nodes) is done at the database level. According to Zafar *et al*., (2016) Neo4j which is a graph-based NoSQL database system are designed for association management but not for query operations. The chief purpose of graph NoSQL is for interconnected. If a user requires the ability to query data as well as build association of data, then the search base database is a better choice. This implies that Neo4j and other graph NoSQL systems are required for special purposes such as social media, internet and construction where graphical representation (images) might be involved.

The research performed tests on individual systems and also integration for different systems which include; MySQL_Cassandra, MySQL _MongoDB, MySQL_Redis, MySQL_Neo4j, MySQL_Cassandra_MongoDB, MySQL_Cassandra_Redis_MongoDB and lastly MySQL_ Cassandra_Redis_Neo4j_MongoDB. The result in Table 4.2 showed that the proposed system; MySQL_Cassandra_Redis_MongoDB outperformed the other systems in terms distribution of data and throughput.

4.2.1 New System Requirements

The new system as outlined in Table 4.1 requires a robust system that is both capable in size and speed to accommodate vast volumes of data.

Hardware Requirements

The minimum hardware requirements for the new hybrid system named ‘NoSQL^2’ are stipulated in Table 4.1:

Table 4.1 New system requirements

Component	**Specification**
Processor:	AMD FX™ Eight-Core Processor 4 GHz
Memory (RAM):	8.00 GB
System Type:	64-bit Operating System, x64-based Processor
Hard Drive:	1TB HDD
Wireless Connectivity:	802.11b/g/n WLAN
Keyboard & Mouse:	External full-size keyboard & Mouse
Display:	14 inch VGA Monitor

Software Requirements

The software development and testing will require;

1) XAMPP Control panel v3.2.2 (Apache distribution containing MySQL, PHP and Perl).
2) MongoDB 3.6.2 2008R2plus SSL (64 bit)
3) Datastax Community Edition (64 bit) 2.1.8
4) Redis – x64 -2.8.2104
5) Neo4j Desktop -1.1.1

4.2.2 Program Development

Choice of Programming Environment

PHP is the Application Layer Controller incorporating MongoDB, Cassandra, Redis and Neo4j drivers. PHP is a server-sided scripting language; it's an abbreviation for Hypertext Preprocessor. It is a very popular programming language used by companies like Facebook, Wikipedia, and Yahoo, to mention a few. PHP powers roughly 75% of the entire web. The reason for choosing this is simply because of its versatility and robustness. The system architecture is shown below.

System Architecture

The system architecture in Figure 4.4 outlines the system framework for both SQL and NoSQL systems. The MySQL system is embedded within the PHP software and as such does not require the driver at the system optimizer level. The various NoSQL systems and their various drivers were analyzes for optimization and load balancing purposes which is the crux of this research.

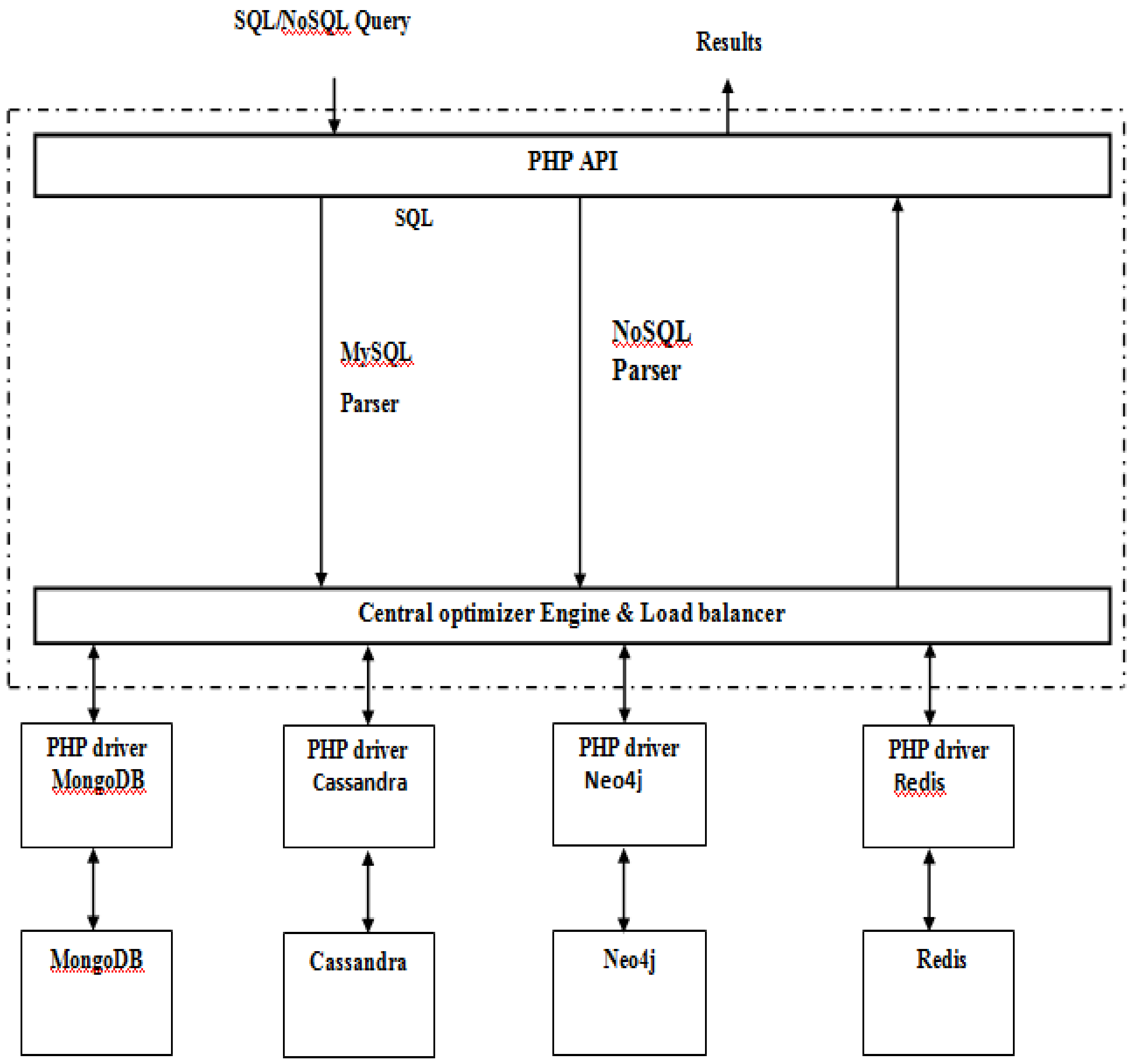


Figure 4.4 System Architecture

Language Justification

The chosen language is PHP which is very fast and has support for multiple database engines.PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. Apache HTTP server is a cross platform which runs the PHP scripting and several other software application. MYSQL database engine is an open source Relational database System, used by many software application because of it stability and relative ease of use. MySQL and PHP are both embedded with the XAMPP control panel which is the Apache distribution for MySQL, PHP and Perl. Due to the open-source nature of MongoDB, Redis, Cassandra and Neo4j, the packages were downloaded online. MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. DataStax is the parent company that owns Cassandra NoSQL database.

As the NoSQL system does not entertain a structured model of data representation, ER diagram rwhich is not only semi-structured but also unstructured. This is because NoSQL does not leverage Entity-Relationship diagram which is obtainable when normalization of data has taken place using foreign keys and join operations to form static schema which are related to one another and run/queried using SQL (Structured Query Language).

4.2.3 System Testing

Test Plan

The black-box testing is a method of software testing that examines the functionality of an application without looking into its internal structures or workings. It is also know as specification-based testing (Khan & Sadiq, 2011).

Test data

The test data was amassed from Phixsoft Global Enterprises Ltd (www.phixsoft.com.ng).

Phixsoft is an importation company managing payment systems; how much has been paid before delivery and the amount left or paid at the collection of good.

Performance Evaluation

Usability tests were carried out on the product by members of staff of Phixsoft Global Enterprises and a client feedback was used as basis for evaluation of success of this product. Implementation of this system is being considered by the company as the tests carried out showed improvement from their current means of onlie trading.

Performance evaluation as shown in Table 4.2 was performed based on integration of seven different systems and evaluation of their capabilities. The systems used for evaluation are; MySQL and Cassandra, MySQL and MongoDB, MySQL and Redis, MySQL and Neo4j, MySQL, Cassandra and MongoDB, MySQL, Cassandra, Redis and MongoDB and lastly MySQL, Cassandra, Redis, Neo4j and MongoDB. Their individual performances based on input and output are tabulated below and figures from the performance of the systems are attached the Sample Outputs in Appendix B.

Table 4.2 Tabular Presentation of Integrated Systems

Systems	Input of 10 Records/completion time in Seconds	Output of 200 Records in seconds
MySQL_Cassandra	6.49026	1.02049
MySQL_MongoDB	1.6133	1.01145
MySQL_Redis	0.79379	1.0441
MySQL_Neo4j (5 records added)	53.06331	1.19218
MySQL_Cassandra_MongoDB	0.73379	1.0154
MySQL_Cassandra_Redis_MongoDB	0.7159	0.01437
MySQL_Cassandra_Redis_Neo4j_MongoDB	11.37879	0.01517

System Validation

The different integrations comprising seven different systems were carried out to check for latency and throughput of the different systems. These tests were carried out for integration of the traditional database system which is MySQL and one of the different NoSQL systems; Redis (Key-value), Cassandra (column-oriented) MongoDB (document store) and Neo4j (graph).

The diagram below is the validation flow diagram (Figure 2.9) by Goyal *et al*., (2016) has the following shortfalls;

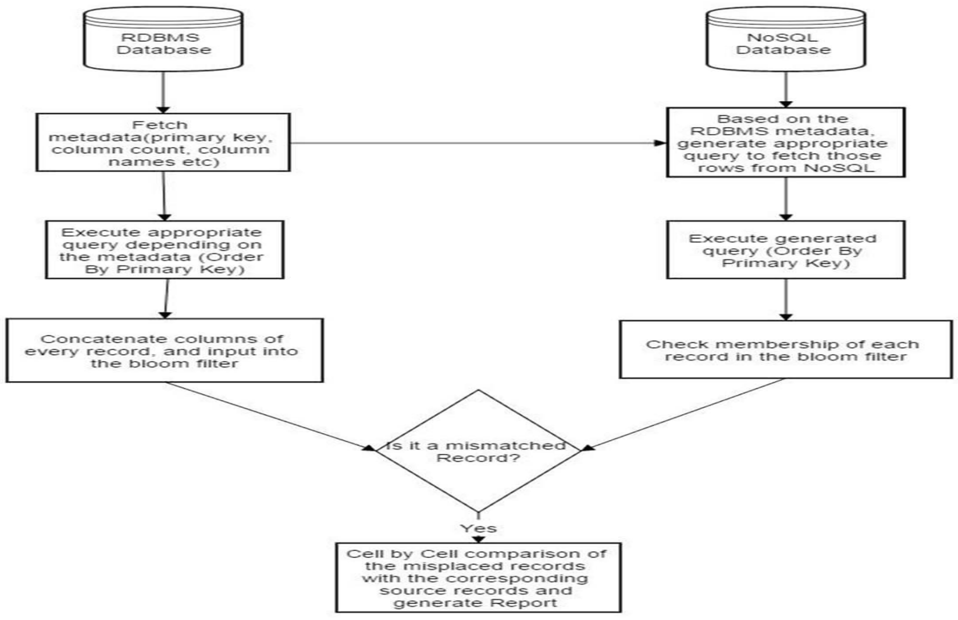
i) The algorithm takes a decision before comparing the data and this leads to loss of data, incorrect records and the probability of false positives.

ii) The Processes should have been completed before a decision is made as to the authenticity of the data in the database.

iii) Non-inclusion of key-value pair NoSQL system by Goyal *et al*., (2016) into the framework.

iv) Incorrect data flow diagram.

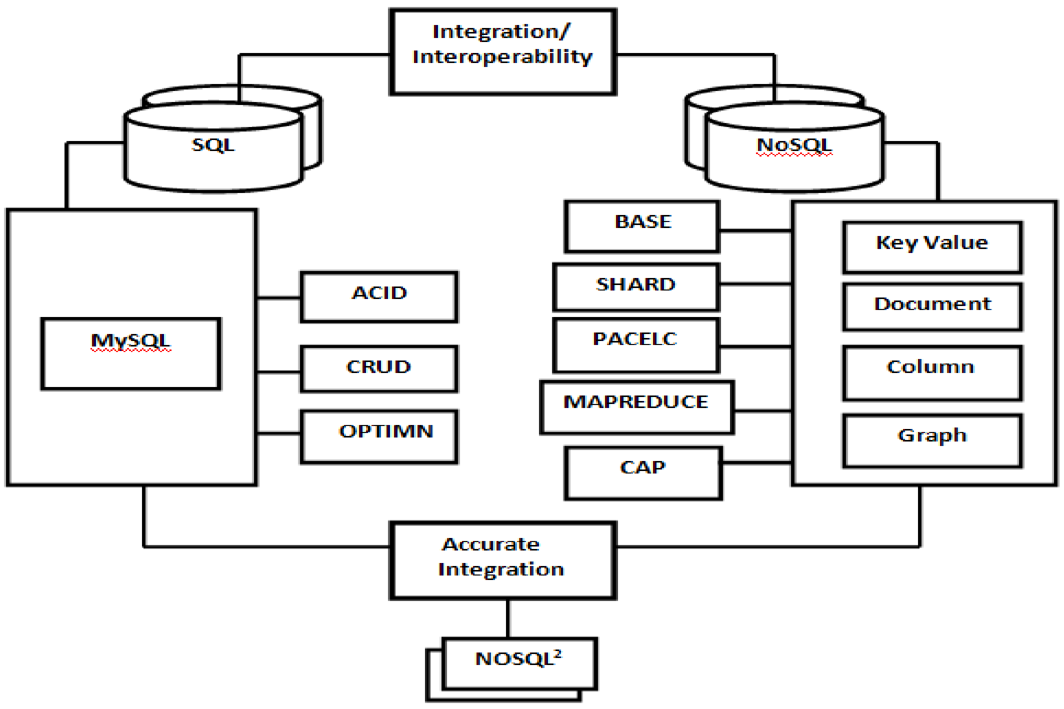
v) Lack of provision of row and column comparisons over the entire data set.


RDBMS Database
NoSQL Database
Fetch metadata(primary key, column count, column names etc)
Based on the RDBMS metadata, generate appropriate query to fetch those rows from NoSQL
Execute appropriate query depending on the metadata (Order By Primary Key)
Execute generated query (Order By Primary Key)
Concatenate columns of every record, and input into the bloom filter
Check membership of each record in the bloom filter
Is it a mismatched Record?
Yes
Cell by Cell comparison of the misplaced records with the corresponding source records and generate Report

The proposed work by Zhang *et al.*, (2011) incorporated the data dictionary of Cassandra; a column-share NoSQL system which implements more write operations than it does read operations and Oracle database; a relational model by affixing a middleware to the architecture. The system by Zhang *et al.*, (2011) has the downside of poor scalability which culminates to a slow system. The existing work by Sanobar and Vanta (2013) performed integration for MySQL and MongoDB.

According to the tests conducted by this research, MySQL and MongoDB are optimal for read-intensive operations and deletion of data. For operations requiring insertions and write-heavy use cases, the MySQL_MongoDB combination will not suffice. The system performed verifications for MySQL and Redis. Redis is a key-value NoSQL system which is superlative for insertions or create operations but is not optimal for other operations inovling intensive read-write functions. The integration of MySQL_Neo4j showed the slowest time as a result of the overhead incurred by the Neo4j. The systems tested showed MySQL_Cassandra_Redis_MongoDB systems to be very viable for different types of operations as the system distributes data based on the strength of the different systems involved. The inclusion of Neo4j; a gragh database NoSQL system to the architecture has a slight overhead as Neo4j is a special purpose system designed for linked or connected datasets which involve complex relationships.

The diagram below is the conceptual (logical) framework involving a new high level model of the system developed (Figure 3.7). As a result of the shortfall of the research by Goyal *et al.*, (2016), the researcher conceived and developed this new high level model to mitigate the inefficiencies observed in the reviewed work. This new high level model ensures accurate integration as row and column insertions were checked to ensure lack of data loss.



Limitations of the system

i) The system developed shows there is an overhead cost when Neo4j graph NoSQL system was integrated alongside Redis, MongoDB and Cassandra. The system throughput is significantly reduced when Noe4j performs query operation and is only needed for special purpose use cases like video or multimedia applications where interconnectedness is highly required.

ii) Research on improving the systems is unavoidable because the systems only communicate by means of the Application Layer which is the PHP. A unified query language that could run on all the attending systems is of great need.

4.2.4 System Conversion

System Changeover involves the smooth migration from one method of operation to another method without disrupting business operation. There are three main methods of system changeover/ conversion and they are; 1) phased implementation whereby a part of the system which requires change is changed and once the transition is fully operational in one area, the lessons learned are used as benchmark to transition the entire system. 2) Another changeover method is the parallel method where the old and new systems run side-by-side, using live examples so that project managers can compare efficieny and reliability of the new system and one the new system is satisfies development requirements, the old system is decommissioned. 3) The last procedure is the direct changeover where system conversion is abrupt because the old system is discarded for an entirely new one.

For the purpose of this research, the Parallel changeover is most suitable as most data is domiciled in the SQL system and integration has been performed for SQL and NoSQL systems. The Original system is MySQL RDBMS which stores only structured data. Read/Write operations are slow as a result of the amount of transactions being run by users which has caused output/access to the information to be slow. As a result of this logjam, a hybrid system comprising of extra three NoSQL systems namely; Redis, MongoDB and Cassandra were employed. The integration also includes MySQL relational system was incorporated into one uniform distributed system in order to spread the data across different distributed systems.

4.2.5 System Security

For the security, the researcher applied:

1. Making sure that only trusted machines can access the database's TCP ports for real life deployment, when the system is run on actual server.
2. Authentication at each database layer (i.e. every single data sent to the database was properly sterilized and escaped). Escaped means avoiding any malicious strings or data being run as part of the original data.
3. Encryption of sensitive database fields. Passwords, server details are all encrypted to avoid revealing them as they are.
4. Keeping unencrypted values in a sandboxed environment (a protected ground where tests can be run freely without incursion/harm to the system).
5. Using sufficient input validation (a two-layer validation was implemented for both the front-end and back-end).

6. Strong user authentication policies for instance commodo positive authentication system. Commodo positive serves as a third-party system that prevents third party from stealing data on the internet. It works as an intrusion-detection system.

4.2.6 Documentation

This research devised a hybrid multifaceted database system for data management. The results showed improvement in performance as a result of seamless sharing/ distribution based on individual system’s capabilities. The system derived is majorly for big businesses and corporations where data management is at the fore of their enterprise. The research employed various tools for the designing this novel database management system and guidelines to installation and running of the systems have been attached to Sample Output in Appendix B of this dissertation.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

The research was primarily embarked on with the aim of providing a data distributed system base on the strengths of the various systems involved. This was achieved by improving on the already existing Iterative and Incremental model. A conceptual framework which consists of the high level model and an architectural layout for exploiting strengths of database management systems were included. Database systems drivers and structures of the management system involving protocol language and parsers were employed for uniform interaction across the database engines and the application layer. A comparative analysis of seven different systems was done and their performance evaluations measured based on qualitative and quantitative analyses.

5.2 Conclusion

A novel methodology 'Holistic Iterative and Incremental Approach for cross-platform modeling' was developed for the design of a cross-platform database system for SQL and NoSQL interoperability. This system showed a better result for managing voluminous data in terms of speed, scalability, continuous availability, and cost effectiveness and location independence.

5.3 Recommendation

Due to the fact that the system eliminates the weaknesses in both database management systems, effectively reduces the development complexity and improves development efficiency of the software systems with multidatabases, it is recommended for storage and management of Big Data in large organizations.

Data management has become a major concern to different facets of people's running of their daily affairs, both in business and the academia and as a result needs to be addressed. The social sciences and arts are not left out in this quest for a robust management system. The researcher recommends that for a start, the institution should create a forum to educate students from their first year in the institution in the field of Data Science and Database management as it will help solve some of their day-to-day need of data management.

This new hybrid system is recommended for use in the Sciences, Social Sciences, Engineering, Medicine, Health Management and Arts.

5.3.1 Application Areas

This research cuts across arts, social science, philosophy, health and also pure and applied sciences. There is growing interest in the field of database management. The areas which this system can be applied are governmental and big non-governmental establishments, hospitals, industries and universities.

5.3.2 Suggestion for Further Research

The research carried out was not exhaustive and requires more work in terms of implementation of the system as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) as the system is still being run on localhost and requires cloud service deployment for commercial purposes.

5.4 Contribution to Knowledge

i) This research contributes an adaptive and deep learning cross-platform system for efficient/effective data management.

ii) The system developed leverages on the benefits/advantages of a data distributed system.

iii) The system also offers better load balancing mechanism (more from the database engines).

REFERENCES

Benefico, S., Gjeci, E., Gomarasca, R.G., Lever, E., Lombardo, S., Ardagna, D. and Di Nitto, E. (2012). Evaluation of the CAP Properties on Amazon SimpleDB and Windows Azure Table Storage. *IEEE 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC).* pp. 430-435.

Boicea, A., Radulescu, F. and Agapin, L. (2012). MongoDB vs Oracle - database comparison. *IEEE 3rd International Conference on Emerging Intelligent Data and Web Technologies.* pp.330—335.

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

Bu, Y., Howe, H., Balazinska, M., and Ernst, M.D. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. *Proceedings of the VLDB Endowment*, Vol. 3. No. 1.

Castelltort, A. and Laurent, A. (2013). Representing history in graph-oriented NoSQL databases: A versioning system. *IEEE 8th International Conference on Digital Information Management (ICDIM).* pp.228-234.

Codd. E.F. (1972). Relational Completeness of Data base Sub Languages. *IBM Research Laboratory, California.*

Cure, O., Kerdjoudj, F., Le Duc., C and Lamolle, M. (2012). On The Potential Integration of an Ontology-Based Data Access Approach in NoSQL Stores. *Third International Conference on Emerging Intelligent Data and Web Technologies.* pp.166-173.

Darwen, H. (2010) An Introduction to Relational Database Theory. *Hugh Darwen & Ventus Publishing ApS. ISBN 978-87-7681-500-4*

Dharmasiri, H.M.L. and Goonetillake, M.D.J.S. (2013). A federated approach on heterogeneous NoSQL data stores. *IEEE International Conference on Advances in ICT for Emerging Regions (ICTer).* pp. 234-239.

Doshi, K. A., Zhong, T., Lu, Z., Tang, X., Lou, T. and Deng, G. (2013). Blending SQL and NewSQL Approaches: Reference Architectures for Enterprise Big Data Challenges.

International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). Pp.163-170.

Ejiofor, V.E., and Okeke, K.K. (2016) Overview of NoSQL and Comparison with SQL Database Management Systems. *The Journal of Computer Science and Its Applications. Vol. 23,No.11.* pp.75-86.

Gadkari, A., Nikam, V.B. and Meshram, B.B. (2014). Implementing Joins over HBASE on Cloud Platform. *IEEE International Conference on Computer and Information (CIT).* pp. 547-554.

Goyal, A., Swaminathan, A., Pande, R., and Attar, V. (2016). Cross Platform (RDBMS to NoSQL) Database Validation Tool Using Bloom Filter. *Fifth International Conference on recent Trends in Information Technology.*

Grolinger, K., Hayes, M., Higashino, W.A., L'Heureux,., Allison, D.S., Capretz, M.A.M. (2014). Challenges for MapReduce in Big Data. *IEEE World Congress on Services (SERVICES).* pp.182-189.

Gudivada, V.N., Rao, D. and Raghavan, V.V. (2014). NoSQL Systems for Big Data Management. *IEEE World Congress on Services (SERVICES).* pp. 190-197.

Gyorodi, C., Gyorodi, R., Pecherle, G., Olah, A. (2015). AComparative Study: MongoDB vs. MySQL. *13th International Conference on Engineering of Modern Electri System (EMES).*pp.1-3.

Han, J., Haihong, E., Le, G. and Du, J. (2011). Survey on NoSQL database. *IEEE 6th International Conference on Pervasive Computing and Applications (ICPCA).* pp.363-366.

Hecht, R. and Jablonski, S. (2011). NoSQL Evaluation: A use case oriented survey. *Proceedings of International Conference on Cloud and Service Computing (CSC).* pp.336-341.

Indrawan-Santiago, M. (2012). Database research: Are we at a crossroad? reflection on NoSQL. *IEEE 15th International Conference on Network-Based Information Systems,* pp.45-51.

Jayathilake, D., Sooriaarachchi, C., Gunawardena, T., Kulasuriya, B. and Dayaratne, T. (2012) A Study Into Capabilities of NoSQL Databases in Handling a Highly Heterogeneous Tree.

IEEE 6th International Conference on Information and Automation for Sustainability (ICIAfS). pp. 106-111.

Jiang, Z., Zheng, Y. and Shi, Y. (2013). Document-Oriented Database-Based Privacy Data Protection Architecture. *IEEE 10th InternationalConference on Web Information System and Application (WISA).* pp. 19-22.

Kanade, A., Gopal, A., Kanade S. (2014). A study of normalization and embedding in MongoDB [C] *IEEE International Conference on Advanced Computing Conference (IACC).*pp.416-421.

Kaur, K. and Rani, R. (2013). Modeling and querying Data in NoSQL Databases. *IEEE International Conference on Big Data*. pp.1-7.

Kepner, J., Gadepally, V., Hutchinson, D., Jananthan, H., Mattson, T., Samsi, S., and Reuther, A. (2016). Associative Array Model of SQL, NoSQL, and NewSQL Databases. *IEEE 2016.*

Khan, M.A. and Sadiq, M. (2011). Analysis of black box software testing techniques: A case study. *IEEE International Conference and Workshop on Current Trends in Information Technology (CTIT)*. Pp. 1-5.

Lawrence, R. (2014). Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB. *International Conference on Computational Science and Computational Intelligence (CSCI).* Pp.433-434

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise?. *IEEEComputing*, 43(2), pp.12-14.

Li C and Gu J. (2019) An integration approach of hybrid databases based on SQL in cloud computing environment. *Softw: Pract Exper*. 2019;49:401–422. https://doi.org/10.1002/spe.2666

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

Liu, Y. and Vitolo, T.M. (2013). Graph Data Warehouse: Steps to Integrating Graph Databases Into the Traditional Conceptual Structure of a Data Warehouse. *IEEE International Congress on Big Data (BigData Congress).* Pp.433-434.

Lombardo, S., Di Nitto, E, Ardagna, D. (2012). Issues in Handling Complex Data Structures with NoSQL databases. *IEEE 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. pp. 443-448.

Lomotey, R.K and Deters, R. (2014). Towards Knowledge Discovery in Big data. *IEEE 8th International Symposium on Service Oriented System Engineering (SOSE).* pp. 181-191.

Neal Ford, 2006, Available:

http://nealford.com/memeagora/2006/12/05/Polyglotprogramming.html

Nyati, S.S., Pawar, S. and Ingle, R. (2013). Performance evaluation of unstructured NoSQL data over distributed framework. *IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI).* pp. 1623-1627.

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

Pettersen, R., Valvag, S., Kvalnes, A. and Johansen, D. (2014). Jovaku: Globally Distributed Caching for Cloud Database Services Using DNS. *IEEE 2nd International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014* pp.127--135.

Qi, M., Liu, Y., Lu, L., Liu, J. and Li, M. (2014). Big Data Management in Digital Forensics. *IEEE 17th International Conference on Computational Science and Engineering (CSE).* Pp.238-243.

Ramanathan, S., Goel, S. and Alagumalai, S. (2011). Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable. *International Conference on Recent Trends in Information Systems (ReTIS) 2011* pp.165--168.

Sanobar, K. and Vanita, M.(2013). SQL Support over MongoDB using Metadata. *International Journal of Scientific and Research Publications, Volume 3, Issue 10.*

Scherzinger, S., Klettke, M., and Storl, U. (2013). Managing Schema Evolution in NoSQL Data Stores. *arXiV preprint arXiv:1308.0515.*

Scott Leberknight, 2008, Available:

http://www.sleberknight.com/blog/sleberkn/entry/polyglot persistence

Srivastava, K. and Shekokar, N. (2016). A Polyglot Persistence Approach for E-Commerce Business Model. *IEEE International Conference on Information Science (ICIS)*pp.7-11.

Stanescu, L., Brezonan, M., and Burdescu, D.D. (2016). Automatic Mapping of MySQL Databases to NoSQL MongoDB. *Proceedings of the Ferated Conference on Computer Science and Information Systems.*pp.837-840.

Sundhara, K., Srividya, and Mohanavalli, S. (2017). A Performance Comparison of Document Oriented NoSQL Databases. *IEEE International Conference on Computer*, *Communication, and Signal Processing (ICCCSP).*

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

Vathy-Fogarassy, A. and Hugyák, T. (2017). Uniform data access platform for SQL and NoSQL database systems. *ELSEVIER Information Systems 69*. Pp 93 -105.

Vilaca, R., Cruz, F., Pereira, J., and Oliveira, R. (2013). An Effective Scalable SQL Engine for NoSQL Databases. *International Federation for Information Processing IFIP*.

Wang, G. and Tang, J. (2012). The NoSQL Principles and Basic Application of Cassandra Model. *IEEE International Conference on Computer Science & Service System (CSSS).* pp.1332-1335.

Wei-ping, Z., Ming-Xin, L. and Huan, C. (2011). Using MongoDB to implement textbook management system instead of MySQL. *IEEE 3rd International Conference on Communication Software and Networks (ICCSN), 2011.* pp.303--305.

Wu, C., Zhu, Q., Zhang, Y., Du, Z., Ye, X., Qin, H., Zhou, Y., (2017). A NoSQL-SQL Hybrid Organization and Management Approach for Real-Time Geospatial Data: A Case Study of Public Security Video Surveillance. *International Journal of Geo-Information.*

Xiang, P., Hou, R. and Zhou, Z. (2010). Cache and consistency in nosql. *IEEE International Conference on Computer Science and Information Technology (ICCSIT).* 6, pp.117-120.

Zafar, R., Yafi, E., Zuhairi, M.F., and Dao, H., (2016). Big Data: The NoSQL and RDBMS review. *International Conference on Information and Communication Technology (ICICTM).*pp120-126.

Zahid, A., Masood, R. and Shibli, M.A. (2014). Security of sharded NoSQL databases: A comparative analysis. *IEEE Conference on Information Assurance and Cyber Security (CIACS).* pp. 1-8.

Zhang, H., Wang, Y. and Han, J. (2011). Middleware design for integrating relational database and NoSQL based on data dictionary. *IEEE International Conference on Transportation, Mechanical and Electrical Engineering (TMEE).* pp. 1469-1472.

Zhao, G., Lin, Q., Li, L. and Li, Z. (2014). Schema Conversion Model of SQL Database to NoSQL Database. *IEEE 9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC).* Pp.355-362.

APPENDIX A

PROGRAM LISTINGS

INDEX.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

getAppHeader("User Records");
getCloseHeader();
getHeaderTag();
getSideBar();

?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        Multiple Database Systems Interpolation
      </h1>
    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <div class="row">
        <div class="col-xs-12">
          <div class="box">
            <div class="box-header">

              <div class="box-tools">
                <div class="input-group input-group-sm" style="width: 150px;">
                  <!--
                  <input type="text" name="table_search" class="form-control pull-right"
placeholder="Search">

                  <div class="input-group-btn">
                    <button type="submit" class="btn btn-default"><i class="fa fa-search"></i></button>


                  </div>
                  -->
                </div>
              </div>
            </div>


            <!-- /.box-header -->
            <div class="box-body table-responsive no-padding">

                    <a href="mysql.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>MySQL System</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>

                    <a href="mongodb.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>MongoDB System</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>

                    <a href="cassandra.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>Cassandra System</h3></span>


                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>




                    <a href="redis.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>Redis System</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>

                    <a href="neo4j.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>Neo4J System</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>

                    <a href="mysql_mongo.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
                            <div class="info-box-content">
                              <span class="info-box-text"><h3>MySQL & MongoDB</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>

                          </div>
                        </div>
                    </a>

                    <a href="mysql_cassandra.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
                            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>


                            <div class="info-box-content">
                              <span class="info-box-text"><h3>MySQL & Cassandra</h3></span>
                              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
                            </div>
                          </div>
                        </div>
                    </a>

                    <a href="mysql_redis.php">
                        <div class="col-md-3 col-sm-6 col-xs-12">
                          <div class="info-box">
```

```
            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
            <div class="info-box-content">
              <span class="info-box-text"><h3>MySQL & Redis</h3></span>
              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
            </div>
          </div>
        </div>
      </a>

      <a href="mysql_neo4j.php">
        <div class="col-md-3 col-sm-6 col-xs-12">
          <div class="info-box">
            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
            <div class="info-box-content">
              <span class="info-box-text"><h3>MySQL & Neo4J</h3></span>
              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
            </div>
          </div>
        </div>
      </a>

      <a href="mysql_cassandra_mongodb.php">
        <div class="col-md-3 col-sm-6 col-xs-12">
          <div class="info-box">
            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
            <div class="info-box-content">
              <span class="info-box-text"><h3>MySQL, Cassandra<br>& MongoDB</h3></span>
              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
            </div>


          </div>
        </div>
      </a>

      <a href="mysql_cassandra_redis_mongodb.php">
        <div class="col-md-3 col-sm-6 col-xs-12">
          <div class="info-box">
            <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
            <div class="info-box-content">

              <span class="info-box-text"><h3>MySQL, Cassandra<br>Redis &
MongoDB</h3></span>
              <!--<span class="info-box-number"><small>Click to run test</small></span>-->
            </div>
          </div>
        </div>
      </a>

      <!--
      <a href="">
```

```
          <div class="col-md-3 col-sm-6 col-xs-12">
            <div class="info-box">
              <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
              <div class="info-box-content">
                <span class="info-box-text"><h3>MySQL, Neo4J<br>& MongoDB</h3></span>
                <span class="info-box-number"><small>Click to run test</small></span>
              </div>
            </div>
          </div>
        </a>

        <a href="">
          <div class="col-md-3 col-sm-6 col-xs-12">
            <div class="info-box">
              <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
              <div class="info-box-content">
                <span class="info-box-text"><h3>MySQL, Cassandra,<br>Redis &
MongoDB</h3></span>
                <span class="info-box-number"><small>Click to run test</small></span>
              </div>
            </div>
          </div>
        </a>



        <a href="">
          <div class="col-md-3 col-sm-6 col-xs-12">
            <div class="info-box">
              <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
              <div class="info-box-content">
                <span class="info-box-text"><h3>MySQL, Cassandra,<br>Neo4J &
MongoDB</h3></span>
                <span class="info-box-number"><small>Click to run test</small></span>
              </div>
            </div>
          </div>
        </a>
        -->
        <a href="mysql_cassandra_redis_neo4j_mongodb">
          <div class="col-md-3 col-sm-6 col-xs-12">
            <div class="info-box">
              <span class="info-box-icon bg-aqua"><i class="ion ion-ios-gear-outline"></i></span>
              <div class="info-box-content">
                <span class="info-box-text"><h3>MySQL, Cassandra,<br>Redis, Neo4J &
MongoDB</h3></span>
                <!--<span class="info-box-number"><small>Click to run test</small></span>-->
              </div>
            </div>
          </div>
        </a>
```

```
            </div>
            <!-- /.box-body -->
          </div>
          <!-- /.box -->
        </div>
      </div>
    </section>
    <!-- /.content -->
  </div>

<?    getPageFooter()?>
```

MONGODB.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

getAppHeader("MongoDB System");
getCloseHeader();
getHeaderTag();
getSideBar();

include("set_limit.php");
$limit = PAGE_LIMIT;

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MongoDB System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>
      <!--
      <ol class="breadcrumb">
        <li><a href="#"><i class="fa fa-dashboard"></i> Home</a></li>
        <li><a href="#">Tables</a></li>
        <li class="active">Simple</li>
      </ol>
    -->

    </section>

    <!-- Main content -->
    <section class="content">



    <!-- /.row -->

      <div class="row">
        <div class="col-xs-12">
          <div class="box">
            <div class="box-header">

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                    <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
<?=$limit?> Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=edit"?>" class="btn btn-
warning">Update <?=$limit?> Records</a>

                    <a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete Records</a>

              </div>
            </div>



            <!-- /.box-header -->
            <br/><br/>
            <div class="box-body table-responsive no-padding">
                <?php

                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                    $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
```

```
		$products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product 6","Product 7","Product 8","Product 9","Product 10","Product 11");
		$admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
		// start script execution timer here
		$starttime = microtime(true);

		//connectAppDB();




		if($opt=="add"){//add records
		   $display_cap = "Add Query: ";
		   try {
		      $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");

		      $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
		      $res = $mng->executeCommand("products", $stats);
		      $stats = current($res->toArray());
		      for($i=0;$i<$limit;$i++){
		         $bulk = new MongoDB\Driver\BulkWrite;
		         $document = [
		                     'id' => new MongoDB\BSON\ObjectId,
		                     'customers' => $customers[rand(0,10)],
		                     'store' => $stores[rand(0,10)],
		                     'product' => $products[rand(0,10)],
		                     'amount' => rand(500,10000),
		                     'addedby' => $admins[rand(0,2)],
		                     'updatefield' => 1,
		                     'createdon' => date("Y-m-d H:i:s")
		         ];
		         $bulk->insert($document);
		         $mng->executeBulkWrite('products.records', $bulk);
		      }

		   } catch (MongoDB\Driver\Exception\Exception $e) {

		      $filename = basename(__FILE__);

		      echo "The $filename script has experienced an error.\n";
		      echo "It failed with the following exception:\n";

		      echo "Exception:", $e->getMessage(), "\n";
		      echo "In file:", $e->getFile(), "\n";
		      echo "On line:", $e->getLine(), "\n";
		      exit;
		   }

		   $endtime= microtime(true);
		}elseif($opt=="edit"){
		   $display_cap = "Update Query: ";
```

```
        $manager = new MongoDB\Driver\Manager('mongodb://localhost:27017');
        for($i=0;$i<$limit;$i++){
            $bulk = new MongoDB\Driver\BulkWrite;
            $bulk->update(
                ['updatefield' => 1],
                ['$set' => ['updatefield' => 3]],
                ['multi' => false, 'upsert' => false]


            );
            $result = $manager->executeBulkWrite('products.records', $bulk);
        }
        $endtime= microtime(true);

        //RESET EVERYTHING AGAIN,
        for($i=0;$i<$limit;$i++){
            $bulk = new MongoDB\Driver\BulkWrite;
            $bulk->update(
                ['updatefield' => 3],
                ['$set' => ['updatefield' => 1]],
                ['multi' => false, 'upsert' => false]
            );
            $result = $manager->executeBulkWrite('products.records', $bulk);
        }

    }elseif($opt=="delete"){
        $display_cap = "Delete Query: ";

        $bulk = new MongoDB\Driver\BulkWrite;
        $bulk->delete([], []);
        $manager = new MongoDB\Driver\Manager('mongodb://localhost:27017');
        $result = $manager->executeBulkWrite('products.records', $bulk);

        $endtime= microtime(true);
    }else{//for read timer
        $display_cap = "Read Query: ";
    }
  ?>

  <table class='table table-hover'>
    <tr>
      <th>#</th>
      <th>Customer</th>
      <th>Store/Branch</th>
      <th>Product</th>
      <th>Amount(&#8358;)</th>
      <th>Added By</th>
      <th>Date</th>
    </tr>
<?php
```

```
        //$customerid = $_REQUEST['customerid'];
        $mongo = new MongoDB\Driver\Manager("mongodb://localhost:27017");
        //$filter      = ['customerid' => $customerid];
        $filter = [];
        $options = [];
        $query = new \MongoDB\Driver\Query($filter, $options);
        $rows   = $mongo->executeQuery('products.records', $query);


        $total_rows=0;
        if(count($rows)>0){
          $n=1;
          foreach ($rows as $document) {
            $total_rows++;
            echo "
              <tr>
                <td>".$n."</td>
                <td>".@$document->customers."</td>
                <td>".@$document->store."</td>
                <td>".@$document->product."</td>
                <td>".@format_number($document->amount)."</td>
                <td>".@$document->addedby."</td>
                <td>".@system_date($document->createdon)."</td>
              </tr>
            ";
            $n++;
          }
        }else{
          echo "<div class='callout callout-info'>Sorry no records has been added yet.</div>";
        }

        if($opt==""){//for read
          $endtime= microtime(true);
        }

        $timediff = $endtime - $starttime;
        //echo $starttime."<br>";
        //echo $endtime."<br>";
        //echo $timediff;//exit;
        $time_elapsed = secondsToTime($timediff);
      ?>
        </table>
      </div>
      <!-- /.box-body -->
     </div>
     <!-- /.box -->
    </div>
   </div>
  </section>
  <!-- /.content -->
</div>
```

```
<?    getPageFooter()?>

<script type="text/javascript">
    $(document).ready(function() {




        $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");

        $('#record_rows').html("[Total Records: <?=$total_rows?>]");


    });    //end of document ready function

    function askDelete(){
        if(confirm("Delete All Records?")){
            parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
        }
    }

</script>
```

MYSQL.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

getAppHeader("MySQL System");
getCloseHeader();
getHeaderTag();
getSideBar();

include("set_limit.php");
$limit = PAGE_LIMIT;

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>



        MySQL System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <div class="row">
        <div class="col-xs-12">
          <div class="box">
            <div class="box-header">

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                    <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
<?=$limit?> Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=edit"?>" class="btn btn-
warning">Update <?=$limit?> Records</a>

                    <a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete Records</a>

              </div>
            </div>



            <!-- /.box-header -->
            <br/><br/>
            <div class="box-body table-responsive no-padding">
                <?php

                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
```

```
          // start script execution timer here
          $starttime = microtime(true);

          connectAppDB();

          if($opt=="add"){//add records
            $display_cap = "Add Query: ";
            for($i=0;$i<$limit;$i++){
              $insertcat = mysql_query("INSERT INTO sales (customers, storeid, productid, amount,
addedby, addedon)
              VALUES
              ('".$customers[rand(0,10)]."','".rand(0,10)."','".rand(0,10)."','".rand(500,10000)."',
'".rand(1,3)."', '".date("Y-m-d H:i:s")."')");
            }
            $endtime= microtime(true);
          }elseif($opt=="edit"){
            $display_cap = "Update Query: ";
            for($i=0;$i<$limit;$i++){
              mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
            }
            $endtime= microtime(true);
            //RESET EVERYTHING AGAIN,
            mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
          }elseif($opt=="delete"){
            $display_cap = "Delete Query: ";
            mysql_query("DELETE FROM sales");
            $endtime= microtime(true);
          }else{//for read timer
            $display_cap = "Read Query: ";
          }


        ?>

        <table class='table table-hover'>
          <tr>
            <th>#</th>
            <th>Customer</th>
            <th>Store/Branch</th>
            <th>Product</th>
            <th>Amount(&#8358;)</th>
            <th>Added By</th>
            <th>Date</th>
          </tr>
      <?php



        $query = "SELECT a.*, b.caption as 'store', c.caption as 'product', d.usernames,
                (SELECT SUM(e.amount) FROM sales e WHERE e.addedby = d.id) AS 'paidnaira'

                FROM sales a, stores b, products c , users d
                WHERE a.storeid = b.id AND a.productid = c.id AND a.addedby = d.id
                ORDER BY id DESC";

            $records = mysql_query($query);
            $total_rows=mysql_num_rows($records);
            if($total_rows>0){
              $n=1;
              while(@$record = mysql_fetch_array($records)){
                echo "
                  <tr>
                    <td>".$n."</td>
                    <td>".$record['customers']."</td>
                    <td>".$record['store']."</td>
                    <td>".$record['product']."</td>
                    <td>".format_number($record['amount'])."</td>
                    <td>".$record['usernames']."</td>
                    <td>".system_date($record['addedon'],2)."</td>
                  </tr>
                ";
                $n++;
              }
            }else{
              echo "<div class='callout callout-info'>Sorry no records has been added yet.</div>";
            }

            if($opt==""){//for read
              $endtime= microtime(true);
            }

            $timediff = $endtime - $starttime;
            //echo $starttime."<br>";
            //echo $endtime."<br>";
            //echo $timediff;//exit;
            $time_elapsed = secondsToTime($timediff);
          ?>
            </table>
          </div>
          <!-- /.box-body -->
        </div>
        <!-- /.box -->
      </div>



    </div>
  </section>
  <!-- /.content -->
</div>
```

```
<?   getPageFooter()?>

<script type="text/javascript">
   $(document).ready(function() {

      $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");

      $('#record_rows').html("[Total Records: <?=$total_rows?>]");


   });   //end of document ready function

   function askDelete(){
      if(confirm("Delete All Records?")){
         parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
      }
   }

</script>
```

NEO4J.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

require_once 'neo4j/graphaware/vendor/autoload.php';
use GraphAware\Neo4j\Client\ClientBuilder,
   GraphAware\Neo4j\OGM\EntityManager;



$client = ClientBuilder::create()
   ->addConnection('default', 'http://neo4j:userpass@localhost:7474') // Example for HTTP connection
configuration (port is optional)
   ->addConnection('bolt', 'bolt://neo4j:userpass@localhost:7687') // Example for BOLT connection
configuration (port is optional)
   ->build();




/*
require('neo4j/vendor/autoload.php');

use Everyman\Neo4j\Client,
   Everyman\Neo4j\Transport,
   Everyman\Neo4j\Node,
   Everyman\Neo4j\Relationship;
```

```
$client = new Everyman\Neo4j\Client(
        (new Everyman\Neo4j\Transport\Curl('localhost',7474))
            ->setAuth('neo4j','userpass')
);


//$keanu = new Node($client);
//$keanu->setProperty('name', 'Keanu Reeves')->save();
$laurence = new Node($client);
//$laurence->setProperty('name', 'Laurence Fishburne')->save();


echo $laurence->getProperty('name') . " was in:\n";

*/




/*
$client = ClientBuilder::create()
    ->addConnection('default', 'http://neo4j:userpass@localhost:7474') // Example for HTTP connection
configuration (port is optional)
    ->addConnection('bolt', 'bolt://neo4j:userpass@localhost:7687') // Example for BOLT connection
configuration (port is optional)
    ->build();

//$client->run('CREATE (n:Person) SET n += {infos}', ['infos' => ['name' => 'Ales', 'age' => 34]]);


$result = $client->run('MATCH (n:Person) RETURN n');
// a result always contains a collection (array) of Record objects

// get all records
//$records = $result->getRecords();


//$query = 'MATCH (n:Person) n, n.name as name, n.age as age';



///$query = $client->run('MATCH (n:Person) RETURN n');
//$result = $client->run($query);


foreach ($result->getRecords() as $record) {
    ?>
    <!--<pre><?//=print_r($record->get('GraphAware\Neo4j\Client\Formatter\Type\Node
Object'))?></pre>-->
    <?
    //print_r($record->get('n')->value('name')); // nodes returned are automatically hydrated to Node
objects
    //echo "<br><br>";

    ///echo $record->value('name');
    echo $record->get('n')->value('age');
    //echo $record->value('age') . PHP_EOL;

        //echo sprintf('Person name is : %s and has %d number of friends', $record->value('name'),
count($record->value('friends')));

}

?>
<!--<pre><?//=print_r($result->getRecords())?></pre>-->
<?

exit;

*/

getAppHeader("Neo4J System");
getCloseHeader();
getHeaderTag();
getSideBar();

//include("set_limit.php");
//$limit = PAGE_LIMIT;
$limit = 50;

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];



}

?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        Neo4J System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>
    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <div class="row">
        <div class="col-xs-12">
          <div class="box">
            <div class="box-header">

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                    <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
<?=$limit?> Records</a>

                    <!--
                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=edit"?>" class="btn btn-
warning">Update <?=$limit?> Records</a>

                    <a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete Records</a>
                    -->
              </div>
            </div>




            <!-- /.box-header -->
            <br/><br/>
            <div class="box-body table-responsive no-padding">
                <?php

                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                    $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
                    $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                    $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                    // start script execution timer here
                    $starttime = microtime(true);
        if($opt=="add"){//add records
            $display_cap = "Add Query: ";
            for($i=0;$i<$limit;$i++){
                $client->run('CREATE (n:Sales) SET n += {infos}',
                    ['infos' =>
                        [
                            'customers' => $customers[rand(0,10)],
                            'stores' => $stores[rand(0,10)],
                            'products' => $products[rand(0,10)],
                            'amount' => rand(500,10000),
                            'addedby' => $admins[rand(0,2)],
                            'addedon' => date("Y-m-d H:i:s"),
                            'updatefield' => 0
                        ]
                    ]
                );
            }
            $endtime= microtime(true);
        }elseif($opt=="edit"){
            $display_cap = "Update Query: ";

            $cluster   = Cassandra::cluster()->build();
            $keyspace  = 'store';//database name
            $session   = $cluster->connect($keyspace);
            $rowz = $session->execute(new Cassandra\SimpleStatement("SELECT * FROM sales
LIMIT ".$limit));
            foreach ($rowz as $rcd) {
                $update = $session->execute(new Cassandra\SimpleStatement




                    ("UPDATE sales SET updatefield = '0' WHERE id = ".$rcd['id'].""));
            }
            $endtime= microtime(true);

            //RESET EVERYTHING AGAIN,
            foreach ($rowz as $rcd) {
                $update = $session->execute(new Cassandra\SimpleStatement
                    ("UPDATE sales SET updatefield = '1' WHERE id = ".$rcd['id'].""));
            }
        }elseif($opt=="delete"){
            $display_cap = "Delete Query: ";

            $endtime= microtime(true);
        }else{//for read timer
            $display_cap = "Read Query: ";
        }
    ?>
```

```
                <table class='table table-hover'>
                    <tr>
                      <th>#</th>
                      <th>Customer</th>
                      <th>Store/Branch</th>
                      <th>Product</th>
                      <th>Amount(&#8358;)</th>
                      <th>Added By</th>
                      <th>Date</th>
                    </tr>
            <?php


                $result = $client->run('MATCH (n:Sales) RETURN n');
// a result always contains a collection (array) of Record objects

// get all records
//$records = $result->getRecords();



//$query = 'MATCH (n:Person) n, n.name as name, n.age as age';
///$query = $client->run('MATCH (n:Person) RETURN n');
//$result = $client->run($query);




            //    if(count($result)>0){
                $n=1;
                $total_rows=0;


                foreach ($result->getRecords() as $record) {
                    $total_rows++;
                    echo "
                        <tr>
                            <td>".$n."</td>
                            <td>".$record->get('n')->value('customers')."</td>
                            <td>".$record->get('n')->value('stores')."</td>
                            <td>".$record->get('n')->value('products')."</td>
                            <td>".format_number($record->get('n')->value('amount'))."</td>
                            <td>".$record->get('n')->value('addedby')."</td>
                            <td>".system_date($record->get('n')->value('addedon'),2)."</td>
                        </tr>
                    ";
                    $n++;
                }
                /*
                }else{
                    echo "<div class='callout callout-info'>Sorry no records has been added yet.</div>";
                }
                */
```

```
        if($opt==""){//for read
          $endtime= microtime(true);
        }

        $timediff = $endtime - $starttime;
        //echo $starttime."<br>";
        //echo $endtime."<br>";
        //echo $timediff;//exit;
        $time_elapsed = secondsToTime($timediff);
      ?>
        </table>
      </div>
      <!-- /.box-body -->
     </div>
     <!-- /.box -->
    </div>
   </div>
  </section>
  <!-- /.content -->
 </div>

<?   getPageFooter()?>

<script type="text/javascript">
  $(document).ready(function() {

    $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");



    $('#record_rows').html("[Total Records: <?=$total_rows?>]");


  });   //end of document ready function

  function askDelete(){
    if(confirm("Delete All Records?")){
      parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
    }
  }

</script>
```

REDIS.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");
include("set_limit.php");
```

```
require "predis/autoload.php";
Predis\Autoloader::register();

getAppHeader("Redis System");
getCloseHeader();
getHeaderTag();
getSideBar();

include("set_limit.php");
$limit = PAGE_LIMIT;

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>



        Redis System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>
    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <div class="row">
        <div class="col-xs-12">
          <div class="box">
            <div class="box-header">

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                    <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>

                    <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add / 
Update <?=$limit?> Records</a>
```

```
                <!--
                <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=edit"?>" class="btn btn-
warning">Update <?=$limit?> Records</a>
                -->
                <a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete Records</a>

            </div>
          </div>



          <!-- /.box-header -->
          <br/><br/>
          <div class="box-body table-responsive no-padding">
             <?php

                $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store


7","Store 8","Store 9","Store 10","Store 11");
                $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                // start script execution timer here
                $starttime = microtime(true);

                if($opt=="add"){//add records
                   $display_cap = "Add Query: ";
                   try {
                      $redis = new Predis\Client();
                      for($i=0;$i<$limit;$i++){
                         $recrd = array(
                            'customers' => $customers[rand(0,10)],
                            'stores' => $stores[rand(0,10)],
                            'products' => $products[rand(0,10)],
                            'amount' => rand(500,10000),
                            'addedby' => $admins[rand(0,2)],
                            'addedon' => date("Y-m-d H:i:s"),
                            'updatefield' => 0
                         );
                         $redis->hmset('record'.$i, $recrd);
                      }
                   }catch (Exception $e) {
                      die($e->getMessage());
                   }
                   $endtime= microtime(true);
                }elseif($opt=="edit"){
                   $display_cap = "Update Query: ";
```

```
                    $endtime= microtime(true);
                }elseif($opt=="delete"){
                    $display_cap = "Delete Query: ";
                    $redis = new Predis\Client();
                    for($i=0;$i<$limit;$i++){
                        @$redis->hdel("record".$i,
array("customers","stores","products","amount","addedby","addedon","updatefield"));
                    }
                    $endtime= microtime(true);
                }else{//for read timer
                    $display_cap = "Read Query: ";
                }
            ?>

            <table class='table table-hover'>
                <tr>
                  <th>#</th>
                  <th>Customer</th>
                  <th>Store/Branch</th>



                  <th>Product</th>
                  <th>Amount(&#8358;)</th>
                  <th>Added By</th>
                  <th>Date</th>
                </tr>
        <?php

            $redis = new Predis\Client();
            $n=1;
            $total_rows=0;
            for($i=0;$i<$limit;$i++){
            //foreach ($result as $record) {
                $record = $redis->hgetall('record'.$i);
                $total_rows++;
                if(isset($record['customers'])){
                    echo "
                        <tr>
                            <td>".$n."</td>
                            <td>".$record['customers']."</td>
                            <td>".$record['stores']."</td>
                            <td>".$record['products']."</td>
                            <td>".format_number($record['amount'])."</td>
                            <td>".$record['addedby']."</td>
                            <td>".system_date($record['addedon'],2)."</td>
                        </tr>
                    ";
                    $n++;
                }
            }
```

```
                if($opt==""){//for read
                    $endtime= microtime(true);
                }

                $timediff = $endtime - $starttime;
                //echo $starttime."<br>";
                //echo $endtime."<br>";
                //echo $timediff;//exit;
                $time_elapsed = secondsToTime($timediff);
              ?>
                </table>
              </div>
              <!-- /.box-body -->
            </div>
            <!-- /.box -->
          </div>
        </div>



    </section>
    <!-- /.content -->
  </div>

<?    getPageFooter()?>

<script type="text/javascript">
    $(document).ready(function() {

        $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");

        $('#record_rows').html("[Total Records: <?=$total_rows?>]");


    });    //end of document ready function

    function askDelete(){
        if(confirm("Delete All Records?")){
            parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
        }
    }

</script>
```

PAGELAYOUT.PHP

```
<? function getAppHeader($title){?>
<!DOCTYPE html>
<html>
<head>
```

```
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title><?=$title?></title>
  <!-- Tell the browser to be responsive to screen width -->
  <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"
name="viewport">
  <!-- Bootstrap 3.3.7 -->
  <link rel="stylesheet" href="bootstrap/dist/css/bootstrap.min.css">
  <!-- Font Awesome -->
  <link rel="stylesheet" href="font-awesome/css/font-awesome.min.css">
  <!-- Ionicons -->
  <link rel="stylesheet" href="Ionicons/css/ionicons.min.css">
  <!-- Theme style -->
  <link rel="stylesheet" href="dist/css/AdminLTE.min.css">



  <!-- AdminLTE Skins. Choose a skin from the css/skins
       folder instead of downloading all of them to reduce the load. -->
  <link rel="stylesheet" href="dist/css/skins/_all-skins.min.css">

  <!-- jQuery 3 -->
    <script src="jquery/dist/jquery.min.js"></script>

  <!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and media queries -->
  <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
  <!--[if lt IE 9]>
  <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
  <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
  <![endif]-->

  <!-- Google Font -->
  <link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,600,700,300italic,400italic,600
italic">
<? }?>
<?    function getCloseHeader(){?>
</head>
<body class="hold-transition skin-blue sidebar-mini">
<div class="wrapper">
<?    }?>
<?    function getHeaderTag(){?>
    <header class="main-header">
    <!-- Logo -->
    <a href="../../index2.html" class="logo">
      <!-- mini logo for sidebar mini 50x50 pixels -->
      <span class="logo-mini"><b>S</b>QL</span>
      <!-- logo for regular state and mobile devices -->
      <span class="logo-lg"><b>SQL, NOSQL</b></span>
    </a>
    <!-- Header Navbar: style can be found in header.less -->
    <nav class="navbar navbar-static-top">
      <!-- Sidebar toggle button-->
      <a href="#" class="sidebar-toggle" data-toggle="push-menu" role="button">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </a>

      <div class="navbar-custom-menu">
        <ul class="nav navbar-nav">
          <!-- Messages: style can be found in dropdown.less-->




          <!-- Notifications: style can be found in dropdown.less -->

          <!-- Tasks: style can be found in dropdown.less -->

          <!-- User Account: style can be found in dropdown.less -->
          <li class="dropdown user user-menu">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown">

              <span class="hidden-xs">Ekene Okeke</span>
            </a>

          </li>
          <!-- Control Sidebar Toggle Button -->

        </ul>
      </div>
    </nav>
  </header>
<?    }?>
<?    function getSideBar(){?>
<!-- Left side column. contains the logo and sidebar -->
  <aside class="main-sidebar">
    <!-- sidebar: style can be found in sidebar.less -->
    <section class="sidebar">
      <!-- Sidebar user panel -->
      <div class="user-panel">
        <div class="pull-left image">
          <BR>
        </div>
        <div class="pull-left info">
          <p>CROSS PLATFORM</p>
          <a href="#"><i class="fa fa-circle text-success"></i> Online</a>
        </div>
      </div>
      <!-- search form
      <form action="#" method="get" class="sidebar-form">
        <div class="input-group">
          <input type="text" name="q" class="form-control" placeholder="Search...">
              <span class="input-group-btn">
                <button type="submit" name="search" id="search-btn" class="btn btn-flat"><i class="fa fa-
search"></i>
                </button>
              </span>
        </div>
      </form>
      -->




      <!-- /.search form -->
      <!-- sidebar menu: : style can be found in sidebar.less -->
      <ul class="sidebar-menu" data-widget="tree">
        <li class="header">MAIN NAVIGATION</li>

        <li><a href="./"><i class="fa fa-table"></i> <span>Dashboard</span></a></li>
        <li><a href="mysql.php"><i class="fa fa-book"></i> <span>MySQL</span></a></li>
        <li><a href="mongodb.php"><i class="fa fa-book"></i> <span>MongoDB</span></a></li>
        <li><a href="cassandra.php"><i class="fa fa-book"></i> <span>Cassandra</span></a></li>
        <li><a href="redis.php"><i class="fa fa-book"></i> <span>Redis</span></a></li>
        <li><a href="neo4j.php"><i class="fa fa-book"></i> <span>Neo4J</span></a></li>

        <li><a href="mysql_mongo.php"><i class="fa fa-book"></i>
<span>MySQL_MongoDB</span></a></li>
        <li><a href="mysql_cassandra.php"><i class="fa fa-book"></i>
<span>MySQL_Cassandra</span></a></li>
        <li><a href="mysql_redis.php"><i class="fa fa-book"></i> <span>MySQL_Redis</span></a></li>
        <li><a href="mysql_neo4j.php"><i class="fa fa-book"></i>
<span>MySQL_Neo4j</span></a></li>

        <li><a href="mysql_cassandra_mongodb.php"><i class="fa fa-book"></i>
<span>MySQL_Cassandra_MongoDB</span></a></li>
        <li><a href="mysql_cassandra_redis_mongodb.php"><i class="fa fa-book"></i>
<span>MySQL_Cassandra_Redis_<br>MongoDB</span></a></li>
        <li><a href="mysql_cassandra_redis_neo4j_mongodb.php"><i class="fa fa-book"></i>
<span>MySQL_Cassandra_Redis_<br>Neo4j_MongoDB</span></a></li>
      </ul>
    </section>
    <!-- /.sidebar -->
  </aside>
<?    }?>
<?    function getPageFooter(){?>
<!-- /.content-wrapper -->
  <footer class="main-footer">
    <div class="pull-right hidden-xs">
      <b>Version</b> 2.4.0
    </div>
    <strong>Final Project: SQL, NoSQL Cross Platform &copy; <?=date("Y")?> .</strong> All rights
    reserved.
  </footer>

  <!-- /.control-sidebar -->
  <!-- Add the sidebar's background. This div must be placed
       immediately after the control sidebar -->



  <div class="control-sidebar-bg"></div>
</div>
<!-- ./wrapper -->

<!-- Bootstrap 3.3.7 -->
<script src="bootstrap/dist/js/bootstrap.min.js"></script>
<!-- Slimscroll -->
<script src="jquery.slimscroll.min.js"></script>
<!-- FastClick -->
<script src="fastclick/lib/fastclick.js"></script>
<!-- AdminLTE App -->
<script src="dist/js/adminlte.min.js"></script>
<!-- AdminLTE for demo purposes -->
<script src="dist/js/demo.js"></script>
</body>
</html>
<?    }?>

<? function callErrorMessage(){?>
    <? if(isset($_SESSION['error']) && !empty($_SESSION['error']) && is_array($_SESSION['error'])){
        //echo "isset";
            if(isset($_SESSION['divborder'])){//on success of query
                $class = "alert alert-success alert-dismissible";
            }elseif(isset($_SESSION['alertstyle'])){// warning caution style
                $class = "alert alert-info alert-dismissible";
            }elseif(isset($_SESSION['infostyle'])){// warning caution style
                $class = "alert alert-info alert-dismissible";
            }else{// regular error
                $class = "alert alert-info alert-dismissible";
            }
        ?>
    <br />
            <div class="<?=$class?>">
                <?php foreach ($_SESSION['error'] as $errors)  { ?>
                        <?php echo ($errors) ?>
                        <?php echo "<br>";?>
                <?php } ?>

                <?
                    unset($_SESSION['error']);
                    if(isset($_SESSION['divborder'])){unset($_SESSION['divborder']);}
                if(isset($_SESSION['alertstyle'])){unset($_SESSION['alertstyle']);}
                if(isset($_SESSION['infostyle'])){unset($_SESSION['infostyle']);}
            ?>
            </div>
    <br />



    <? }?>
<? }?>
```

MYSQL_REDIS.PHP

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

require "predis/autoload.php";
Predis\Autoloader::register();

getAppHeader("MySQL - Redis");
getCloseHeader();
getHeaderTag();
getSideBar();

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

    $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL - Redis System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">
    <!-- /.row -->

     <? callErrorMessage();?>

     <div class="row">
       <div class="col-xs-12">
         <div class="box">

           <div class="box-header">
             <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

             <span class="btn-info" id="record_rows"></span>

             <div class="box-tools">

               <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
               <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
               <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->
             </div>
           </div>

           <?php
                   $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                   $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
                   $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                   $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                   // start script execution timer here

                   //DATA FOR MYSQL, USER INFORMATION
                   $gender = array("Male","Female");
                   $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
                   $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");
```

```
		// start script execution timer here
		$starttime = microtime(true);

		//CONNECT TO MYSQL
		connectAppDB();

		if($opt=="add"){//add records
		   $display_cap = "Add Query: ";
		   //echo "exask";exit;
		   for($i=0;$i<10;$i++){
		      $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone, 
salestype, createon)
		      VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."', 'redis', 
'".date("Y-m-d H:i:s")."')");

		      $cusid = (int) mysql_insert_id();

		      if($insertcat){
		         $redis = new Predis\Client();
		         for($e=0;$e<200;$e++){
		            $recrd = array(
		               'customerid' => $cusid,
		               'customers' => $customers[rand(0,10)],
		               'stores' => $stores[rand(0,10)],
		               'products' => $products[rand(0,10)],
		               'amount' => rand(500,10000),
		               'addedby' => $admins[rand(0,2)],
		               'addedon' => date("Y-m-d H:i:s"),
		               'updatefield' => 0
		            );
		            $redis->hmset('record'.$cusid, $recrd);
		         }
		      }
		   }
		   $endtime= microtime(true);
		}elseif($opt=="edit"){
		   $display_cap = "Update Query: ";
		   for($i=0;$i<$limit;$i++){
		      mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
		   }
		   $endtime= microtime(true);
		   //RESET EVERYTHING AGAIN,
		   mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");



		}elseif($opt=="delete"){
		   $display_cap = "Delete Query: ";
		   mysql_query("DELETE FROM sales");
```

```
                    $endtime= microtime(true);
                }else{//for read timer
                    $display_cap = "Read Query: ";
                }
            ?>

          <!-- /.box-header -->
          <div class="box-body table-responsive no-padding">
            <table class="table table-hover">
              <tr>
                <th>#</th>
                <th>Customers</th>
                <th>Gender</th>
                <th>Phone</th>
                <th>Email</th>
                <th>Action</th>
              </tr>

              <?php
                connectAppDB();
                $myquery = "SELECT * FROM users WHERE salestype = 'redis' ORDER BY id DESC";
                $list = mysql_query($myquery);
                $total_rows = mysql_num_rows($list);
                //exit;
                if($total_rows > 0){
                    $i=1;
                    while($record = mysql_fetch_array($list)){
              ?>
                    <tr>
                        <td><?=$i?></td>
                        <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                        <td><?=$record['gender']?></td>
                        <td><?=$record['phone']?></td>
                        <td><?=$record['email']?></td>
                        <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                        <td>
                            <a class="label label-info"
                            onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                        </td>
                    </tr>
              <?php
                        $i++;



                    }
                }
                if($opt==""){//for read
                    $endtime= microtime(true);
                }
                $timediff = $endtime - $starttime;
                $time_elapsed = secondsToTime($timediff);
              ?>
            </table>
          </div>
          <!-- /.box-body -->
        </div>
        <!-- /.box -->
      </div>
    </div>
  </section>
  <!-- /.content -->
</div>


<!--MODAL FORM FOR ADDING RECORDS TO DATABASE -->
  <form name="new_record" action="controller.php" method="POST">
      <div class="modal fade" id="modal-default5">
        <div class="modal-dialog">
          <div class="modal-content">
            <div class="modal-header">
              <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                <span aria-hidden="true">&times;</span></button>
              <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From Redis NoSQl
Database</h4>
            </div>
            <div class="modal-body">

              <div class="box-body">

                  <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                  <div id="show_display"></div>
              </div>

            </div>
            <div class="modal-footer">
              <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
            </div>
          </div>
          <!-- /.modal-content -->



        </div>
        <!-- /.modal-dialog -->
      </div>
  </form>

  <script type="text/javascript">
      $(document).ready(function() {
          $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
          $('#record_rows').html("[Total Records: <?=$total_rows?>]");
      });    //end of document ready function
      function askDelete(){
          if(confirm("Delete All Records?")){
              parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
          }
      }
      function callModalSales(ith){
          $("#modal-default5").modal("show");
          $("#customerr").html($("#customer"+ith).html());
          $("#show_display").html("");
          $('#txtHint').show();
          var loadUrl = "controller.php?action=view_redis_sales&customerid="+ith;
          $.ajax({
              type: "POST",
              url: loadUrl,
              cache: false,
              success: function(output){
                  $("#show_display").html(output);
                  $('#txtHint').hide();
              },
              error: function(xhr, textStatus, errorThrown) {
                  $('#txtHint').hide();
              }
          });
      }
  </script>
<?    getPageFooter()?>
```

MYSQL_NEO4J

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

require_once 'neo4j/graphaware/vendor/autoload.php';



use GraphAware\Neo4j\Client\ClientBuilder,
    GraphAware\Neo4j\OGM\EntityManager;



$client = ClientBuilder::create()
    ->addConnection('default', 'http://neo4j:userpass@localhost:7474') // Example for HTTP connection
configuration (port is optional)
    ->addConnection('bolt', 'bolt://neo4j:userpass@localhost:7687') // Example for BOLT connection
configuration (port is optional)
    ->build();
```

```
getAppHeader("MySQL - Neo4J");
getCloseHeader();
getHeaderTag();
getSideBar();

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
   $opt = $_GET['opt'];
}

   $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL - Neo4J System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <? callErrorMessage();?>




      <div class="row">
        <div class="col-xs-12">
          <div class="box">

            <div class="box-header">
              <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
                <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
```

```
Records</a>
            <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->
          </div>
        </div>

        <?php
            $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
            $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
            $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
            $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
            // start script execution timer here

            //DATA FOR MYSQL, USER INFORMATION
            $gender = array("Male","Female");
            $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
            $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

            // start script execution timer here
            $starttime = microtime(true);

            //CONNECT TO MYSQL
            connectAppDB();

            if($opt=="add"){//add records



              $display_cap = "Add Query: ";
              //echo "exask";exit;
              for($i=0;$i<5;$i++){
                $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
                VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."', 'neo4j',
'".date("Y-m-d H:i:s")."')");

                $cusid = (int) mysql_insert_id();

                if($insertcat){
                  for($e=0;$e<10;$e++){
                    $client->run('CREATE (n:Sales) SET n += {infos}',
                    ['infos' =>
                        [
                            'customers' => $customers[rand(0,10)],
                            'customerid' => $cusid,
                            'stores' => $stores[rand(0,10)],
                            'products' => $products[rand(0,10)],
                            'amount' => rand(500,10000),
                            'addedby' => $admins[rand(0,2)],
                            'addedon' => date("Y-m-d H:i:s"),
                            'updatefield' => 0
                        ]
                    ]
                );
            }
        }
    }
    $endtime= microtime(true);
}elseif($opt=="edit"){
    $display_cap = "Update Query: ";
    for($i=0;$i<$limit;$i++){
        mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
    }
    $endtime= microtime(true);
    //RESET EVERYTHING AGAIN,
    mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
}elseif($opt=="delete"){
    $display_cap = "Delete Query: ";
    mysql_query("DELETE FROM sales");
    $endtime= microtime(true);
}else{//for read timer



    $display_cap = "Read Query: ";
}
?>

<!-- /.box-header -->
<div class="box-body table-responsive no-padding">
  <table class="table table-hover">
    <tr>
      <th>#</th>
      <th>Customers</th>
      <th>Gender</th>
      <th>Phone</th>
      <th>Email</th>
      <th>Action</th>
    </tr>

    <?php
      connectAppDB();
```

```
                $myquery = "SELECT * FROM users WHERE salestype = 'neo4j' ORDER BY id DESC";
                $list = mysql_query($myquery);
                $total_rows = mysql_num_rows($list);
                //exit;
                if($total_rows > 0){
                    $i=1;
                    while($record = mysql_fetch_array($list)){
            ?>
                    <tr>
                        <td><?=$i?></td>
                        <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                        <td><?=$record['gender']?></td>
                        <td><?=$record['phone']?></td>
                        <td><?=$record['email']?></td>
                        <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                        <td>
                            <a class="label label-info"
                            onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                        </td>
                    </tr>
            <?php
                        $i++;
                    }
                }
                if($opt==""){//for read




                    $endtime= microtime(true);
                }
                $timediff = $endtime - $starttime;
                $time_elapsed = secondsToTime($timediff);
            ?>
          </table>
        </div>
        <!-- /.box-body -->
      </div>
      <!-- /.box -->
    </div>
  </div>
 </section>
 <!-- /.content -->
</div>


<!--MODAL FORM FOR ADDING RECORDS TO DATABASE -->
 <form name="new_record" action="controller.php" method="POST">
    <div class="modal fade" id="modal-default5">
      <div class="modal-dialog">
```

```
            <div class="modal-content">
              <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                  <span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From Neo4J
Database</h4>
              </div>
              <div class="modal-body">

                <div class="box-body">

                    <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                    <div id="show_display"></div>
                </div>

              </div>
              <div class="modal-footer">
                <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
              </div>
            </div>
            <!-- /.modal-content -->
          </div>
          <!-- /.modal-dialog -->



        </div>
    </form>

    <script type="text/javascript">
        $(document).ready(function() {
            $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
            $('#record_rows').html("[Total Records: <?=$total_rows?>]");
        });    //end of document ready function
        function askDelete(){
            if(confirm("Delete All Records?")){
                parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
            }
        }
        function callModalSales(ith){
            $("#modal-default5").modal("show");
            $("#customerr").html($("#customer"+ith).html());
            $("#show_display").html("");
            $('#txtHint').show();
            var loadUrl = "controller.php?action=view_neo4j_sales&customerid="+ith;
            $.ajax({
                type: "POST",
                url: loadUrl,
                cache: false,
                success: function(output){
                    $("#show_display").html(output);
                    $('#txtHint').hide();
                },
                error: function(xhr, textStatus, errorThrown) {
                    $('#txtHint').hide();
                }
            });
        }
    </script>
<?    getPageFooter()?>
```

MYSQL_MONGODB

```
<?php
require_once("functions.php");
require_once("pagelayout.php");





getAppHeader("MySQL - MongoDB");
getCloseHeader();
getHeaderTag();
getSideBar();

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

    $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL -  MongoDB System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">
  <!-- /.row -->

    <? callErrorMessage();?>

    <div class="row">
      <div class="col-xs-12">
        <div class="box">

          <div class="box-header">
            <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

            <span class="btn-info" id="record_rows"></span>

            <div class="box-tools">



              <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
              <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
              <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->
            </div>
          </div>

          <?php

                    //DATA FOR MONGODB, SALES INFORMATION
                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                    $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
                    $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                    $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                    // start script execution timer here

                    //DATA FOR MYSQL, USER INFORMATION
                    $gender = array("Male","Female");
                    $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
                    $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

                    // start script execution timer here
                    $starttime = microtime(true);
                    //CONNECT TO MYSQL
                    connectAppDB();

                    //CONNECT TO MONGODB
                    $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");
                    $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
                    $res = $mng->executeCommand("products", $stats);
                    $stats = current($res->toArray());

                    if($opt=="add"){//add records
                       $display_cap = "Add Query: ";
                       //echo "exask";exit;
                       for($i=0;$i<10;$i++){




                          $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
                          VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."',
'mongodb', '".date("Y-m-d H:i:s")."')");

                          $cusid = (int) mysql_insert_id();

                          if($insertcat){
                             for($e=0;$e<200;$e++){
                                $bulk = new MongoDB\Driver\BulkWrite;
                                $document = [
                                        'id' => new MongoDB\BSON\ObjectId,
                                        'customerid' => $cusid,
                                        'customers' => $customers[rand(0,10)],
                                        'store' => $stores[rand(0,10)],
                                        'product' => $products[rand(0,10)],
                                        'amount' => rand(500,10000),
                                        'addedby' => $admins[rand(0,2)],
                                        'updatefield' => 1,
                                        'createdon' => date("Y-m-d H:i:s")
                                ];
                                $bulk->insert($document);
                                $mng->executeBulkWrite('products.records', $bulk);
                             }
                          }
                       }
                       $endtime= microtime(true);
                    }elseif($opt=="edit"){
                       $display_cap = "Update Query: ";
                       for($i=0;$i<$limit;$i++){
                          mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
                       }
```

```
                    $endtime= microtime(true);
                    //RESET EVERYTHING AGAIN,
                    mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
                }elseif($opt=="delete"){
                    $display_cap = "Delete Query: ";
                    mysql_query("DELETE FROM sales");
                    $endtime= microtime(true);
                }else{//for read timer
                    $display_cap = "Read Query: ";






                }
            ?>

          <!-- /.box-header -->
          <div class="box-body table-responsive no-padding">
            <table class="table table-hover">
              <tr>
                <th>#</th>
                <th>Customers</th>
                <th>Gender</th>
                <th>Phone</th>
                <th>Email</th>
                <th>Action</th>
              </tr>

              <?php
                connectAppDB();
                $myquery = "SELECT * FROM users WHERE salestype = 'mongodb' ORDER BY id
DESC";
                $list = mysql_query($myquery);
                $total_rows = mysql_num_rows($list);
                //exit;
                if($total_rows > 0){
                    $i=1;
                    while($record = mysql_fetch_array($list)){
              ?>
                    <tr>
                        <td><?=$i?></td>
                        <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                        <td><?=$record['gender']?></td>
                        <td><?=$record['phone']?></td>
                        <td><?=$record['email']?></td>
                        <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                        <td>
                            <a class="label label-info"
                            onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                    </td>
                  </tr>
              <?php
                    $i++;
                  }
                }
                if($opt==""){//for read
                  $endtime= microtime(true);
                }
                $timediff = $endtime - $starttime;



                $time_elapsed = secondsToTime($timediff);
              ?>
            </table>
          </div>
          <!-- /.box-body -->
        </div>
        <!-- /.box -->
      </div>
    </div>
  </section>
  <!-- /.content -->
</div>


<!--MODAL FORM FOR ADDING RECORDS TO MONGODB DATABASE -->
  <form name="new_record" action="controller.php" method="POST">
    <div class="modal fade" id="modal-default5">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
              <span aria-hidden="true">&times;</span></button>
            <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From MongoDB
NoSQl Database</h4>
          </div>
          <div class="modal-body">

            <div class="box-body">

                <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                <div id="show_display"></div>
            </div>

          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
          </div>
```

```
            </div>
            <!-- /.modal-content -->
          </div>
          <!-- /.modal-dialog -->
        </div>
    </form>

    <script type="text/javascript">



        $(document).ready(function() {
            $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
            $('#record_rows').html("[Total Records: <?=$total_rows?>]");
        });    //end of document ready function
        function askDelete(){
            if(confirm("Delete All Records?")){
                parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
            }
        }
        function callModalSales(ith){
            $("#modal-default5").modal("show");
            $("#customerr").html($("#customer"+ith).html());
            $("#show_display").html("");
            $('#txtHint').show();
            var loadUrl = "controller.php?action=view_mongo_sales&customerid="+ith;
            $.ajax({
                type: "POST",
                url: loadUrl,
                cache: false,
                success: function(output){
                    $("#show_display").html(output);
                    $('#txtHint').hide();
                },
                error: function(xhr, textStatus, errorThrown) {
                    $('#txtHint').hide();
                }
            });
        }
    </script>
<?    getPageFooter()?>
```

MYSQL_CASSANDRA

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

getAppHeader("MySQL - Cassandra");
getCloseHeader();
```

```
getHeaderTag();
getSideBar();

$time_elapsed = "0";



$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
   $opt = $_GET['opt'];
}

   $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL - Cassandra System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <? callErrorMessage();?>

      <div class="row">
        <div class="col-xs-12">
          <div class="box">

            <div class="box-header">
              <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
                <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
                <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
```

```
Records</a>-->
        </div>
      </div>

      <?php
          $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
          $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
          $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
          $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
          // start script execution timer here

          //DATA FOR MYSQL, USER INFORMATION
          $gender = array("Male","Female");
          $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
          $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

          // start script execution timer here
          $starttime = microtime(true);

          //CONNECT TO MYSQL
          connectAppDB();

          if($opt=="add"){//add records
            $display_cap = "Add Query: ";
            //echo "exask";exit;
            for($i=0;$i<10;$i++){
              $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
              VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."',
'cassandra', '".date("Y-m-d H:i:s")."')");

              $cusid = (int) mysql_insert_id();

              if($insertcat){
                $cluster   = Cassandra::cluster()->build();
                $keyspace  = 'store';//database name
```

```
                        $session   = $cluster->connect($keyspace);
                        for($e=0;$e<200;$e++){
                            $statement = $session->execute(new Cassandra\SimpleStatement(
                                "INSERT INTO sales (id, customerid, customers, stores, products, amount,
addedby, addedon, updatefield)
                                VALUES

(now(),'".$cusid."','".$customers[rand(0,10)]."','".$stores[rand(0,10)]."','".$products[rand(0,10)]."','".rand(
500,10000)."', '".$admins[rand(0,2)]."', '".date("Y-m-d H:i:s")."','0')"
                              )
                            );
                        }
                    }
                }
                $endtime= microtime(true);
            }elseif($opt=="edit"){
                $display_cap = "Update Query: ";
                for($i=0;$i<$limit;$i++){
                    mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
                }
                $endtime= microtime(true);
                //RESET EVERYTHING AGAIN,
                mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
            }elseif($opt=="delete"){
                $display_cap = "Delete Query: ";
                mysql_query("DELETE FROM sales");
                $endtime= microtime(true);
            }else{//for read timer
                $display_cap = "Read Query: ";
            }
        ?>

      <!-- /.box-header -->
      <div class="box-body table-responsive no-padding">
        <table class="table table-hover">
          <tr>
            <th>#</th>
            <th>Customers</th>
            <th>Gender</th>
            <th>Phone</th>
            <th>Email</th>
            <th>Action</th>
          </tr>






          <?php
            connectAppDB();
```

```
                    $myquery = "SELECT * FROM users WHERE salestype = 'cassandra' ORDER BY id
DESC";
                    $list = mysql_query($myquery);
                    $total_rows = mysql_num_rows($list);
                    //exit;
                    if($total_rows > 0){
                        $i=1;
                        while($record = mysql_fetch_array($list)){
                ?>
                        <tr>
                            <td><?=$i?></td>
                            <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                            <td><?=$record['gender']?></td>
                            <td><?=$record['phone']?></td>
                            <td><?=$record['email']?></td>
                            <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                            <td>
                                <a class="label label-info"
                                onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                            </td>
                        </tr>
                <?php
                            $i++;
                        }
                    }
                    if($opt==""){//for read
                        $endtime= microtime(true);
                    }
                    $timediff = $endtime - $starttime;
                    $time_elapsed = secondsToTime($timediff);
                ?>
              </table>
            </div>
            <!-- /.box-body -->
          </div>
          <!-- /.box -->
        </div>
      </div>
    </section>
    <!-- /.content -->
  </div>




  <!--MODAL FORM FOR ADDING RECORDS TO DATABASE -->
    <form name="new_record" action="controller.php" method="POST">
        <div class="modal fade" id="modal-default5">
```

```
          <div class="modal-dialog">
            <div class="modal-content">
              <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                  <span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From Cassandra
NoSQl Database</h4>
              </div>
              <div class="modal-body">

                <div class="box-body">

                    <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                    <div id="show_display"></div>
                </div>

              </div>
              <div class="modal-footer">
                <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
              </div>
            </div>
            <!-- /.modal-content -->
          </div>
          <!-- /.modal-dialog -->
        </div>
    </form>

    <script type="text/javascript">
        $(document).ready(function() {
            $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
            $('#record_rows').html("[Total Records: <?=$total_rows?>]");
        });    //end of document ready function
        function askDelete(){
            if(confirm("Delete All Records?")){
                parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
            }
        }
        function callModalSales(ith){
            $("#modal-default5").modal("show");
            $("#customerr").html($("#customer"+ith).html());
            $("#show_display").html("");
            $('#txtHint').show();



            var loadUrl = "controller.php?action=view_cassandra_sales&customerid="+ith;
            $.ajax({
                type: "POST",
                url: loadUrl,
                cache: false,
                success: function(output){
                    $("#show_display").html(output);
                    $('#txtHint').hide();
                },
                error: function(xhr, textStatus, errorThrown) {
                    $('#txtHint').hide();
                }
            });
        }
    </script>
<?    getPageFooter()?>
```

MYSQL_CASSANDRA_MONGO

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

getAppHeader("MySQL - Cassandra - MongoDB");
getCloseHeader();
getHeaderTag();
getSideBar();

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

    $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->



    <section class="content-header">
      <h1>
        MySQL - Cassandra - MongoDB System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <? callErrorMessage();?>

      <div class="row">
        <div class="col-xs-12">
          <div class="box">

            <div class="box-header">
              <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
                <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
                <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->
              </div>
            </div>

            <?php

                    //DATA FOR MONGODB, SALES INFORMATION
                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                    $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
                    $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");



                    $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                    // start script execution timer here

                    //DATA FOR MYSQL, USER INFORMATION
                    $gender = array("Male","Female");
                    $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
                    $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

                    // start script execution timer here
                    $starttime = microtime(true);

                    //CONNECT TO MYSQL
                    connectAppDB();

                    //CONNECT TO MONGODB
                    $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");
                    $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
                    $res = $mng->executeCommand("products", $stats);
                    $stats = current($res->toArray());

                    if($opt=="add"){//add records
                       $display_cap = "Add Query: ";
                       //echo "exask";exit;
                       for($i=0;$i<10;$i++){
                          $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
                          VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."',
'mongodb_cassandra', '".date("Y-m-d H:i:s")."')");
                          $cusid = (int) mysql_insert_id();

                          //NEXT INSERT INTO Cassandra
                          $cluster   = Cassandra::cluster()->build();
                          $keyspace  = 'store';//database name
                          $session   = $cluster->connect($keyspace);
                          $session->execute(new Cassandra\SimpleStatement(
                                "INSERT INTO table_store (id, caption) VALUES
(now(),'".$stores[rand(0,10)]."')"
                         )
                          );




                          //NEXT INSERT INTO MongoDB
                          if($insertcat){
                         for($e=0;$e<200;$e++){
                                $bulk = new MongoDB\Driver\BulkWrite;
                                $document = [
                                            'id' => new MongoDB\BSON\ObjectId,
                                            'customerid' => $cusid,
                                            'customers' => $customers[rand(0,10)],
                                            'store' => $stores[rand(0,10)],
                                            'product' => $products[rand(0,10)],
                                            'amount' => rand(500,10000),
                                            'addedby' => $admins[rand(0,2)],
                                            'updatefield' => 1,
                                            'createdon' => date("Y-m-d H:i:s")
```

```
                    ];
                    $bulk->insert($document);
                    $mng->executeBulkWrite('products.records', $bulk);
                }
            }
        }
        $endtime= microtime(true);
    }elseif($opt=="edit"){
        $display_cap = "Update Query: ";
        for($i=0;$i<$limit;$i++){
            mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
        }
        $endtime= microtime(true);
        //RESET EVERYTHING AGAIN,
        mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
    }elseif($opt=="delete"){
        $display_cap = "Delete Query: ";
        mysql_query("DELETE FROM sales");
        $endtime= microtime(true);
    }else{//for read timer
        $display_cap = "Read Query: ";
    }
?>

<!-- /.box-header -->
<div class="box-body table-responsive no-padding">
  <table class="table table-hover">
    <tr>
      <th>#</th>
      <th>Customers</th>
      <th>Gender</th>



      <th>Phone</th>
      <th>Email</th>
      <th>Added On</th>
      <th>Action</th>
    </tr>

    <?php
        connectAppDB();
        $myquery = "SELECT * FROM users WHERE salestype = 'mongodb_cassandra' ORDER
BY id DESC";
        $list = mysql_query($myquery);
        $total_rows = mysql_num_rows($list);
        //exit;
        if($total_rows > 0){
            $i=1;
            while($record = mysql_fetch_array($list)){
    ?>
```

```
                        <tr>
                            <td><?=$i?></td>
                            <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                            <td><?=$record['gender']?></td>
                            <td><?=$record['phone']?></td>
                            <td><?=$record['email']?></td>
                            <td><?=system_date($record['createon'],2)?></td>
                            <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                            <td>
                                <a class="label label-info"
                                onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                            </td>
                        </tr>
                <?php
                            $i++;
                        }
                    }
                    if($opt==""){//for read
                        $endtime= microtime(true);
                    }
                    $timediff = $endtime - $starttime;
                    $time_elapsed = secondsToTime($timediff);
                ?>
              </table>
            </div>
            <!-- /.box-body -->
          </div>
          <!-- /.box -->
        </div>



      </div>
    </section>
    <!-- /.content -->
  </div>


  <!--MODAL FORM FOR ADDING RECORDS TO MONGODB DATABASE -->
    <form name="new_record" action="controller.php" method="POST">
        <div class="modal fade" id="modal-default5">
          <div class="modal-dialog">
            <div class="modal-content">
              <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                  <span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From
Cassandra/MongoDB NoSQl Database</h4>
              </div>
              <div class="modal-body">
                <div class="box-body">

                    <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                    <div id="show_display"></div>
                </div>

              </div>
              <div class="modal-footer">
                <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
              </div>
            </div>
            <!-- /.modal-content -->
          </div>
          <!-- /.modal-dialog -->
        </div>
    </form>

    <script type="text/javascript">
        $(document).ready(function() {
            $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
            $('#record_rows').html("[Total Records: <?=$total_rows?>]");
        });    //end of document ready function
        function askDelete(){
            if(confirm("Delete All Records?")){




                parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
            }
        }
        function callModalSales(ith){
            $("#modal-default5").modal("show");
            $("#customerr").html($("#customer"+ith).html());
            $("#show_display").html("");
            $('#txtHint').show();
            var loadUrl = "controller.php?action=view_mongo_sales&customerid="+ith;
            $.ajax({
                type: "POST",
                url: loadUrl,
                cache: false,
                success: function(output){
                    $("#show_display").html(output);
                    $('#txtHint').hide();
                },
                error: function(xhr, textStatus, errorThrown) {
                    $('#txtHint').hide();
                }
            });
        }
    </script>
<?    getPageFooter()?>
```

MYSQL_CASSANDRA_REDIS_MONGO

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

require "predis/autoload.php";
Predis\Autoloader::register();


getAppHeader("MySQL - Cassandra - Redis - MongoDB");
getCloseHeader();
getHeaderTag();
getSideBar();







$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
}

    $limit = 50;
?>

  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL - Cassandra - Redis - MongoDB System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->
    <? callErrorMessage();?>

    <div class="row">
      <div class="col-xs-12">
        <div class="box">

          <div class="box-header">
            <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

            <span class="btn-info" id="record_rows"></span>

            <div class="box-tools">

              <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
              <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
              <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->



            </div>
          </div>

          <?php

                //DATA FOR MONGODB, SALES INFORMATION
                $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store 7","Store
8","Store 9","Store 10","Store 11");
                $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                // start script execution timer here

                //DATA FOR MYSQL, USER INFORMATION
                $gender = array("Male","Female");
                $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
                $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

                // start script execution timer here
                $starttime = microtime(true);

                //CONNECT TO MYSQL
                connectAppDB();

                //CONNECT TO MONGODB
                $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");
                $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
                $res = $mng->executeCommand("products", $stats);
                $stats = current($res->toArray());

                if($opt=="add"){//add records
                   $display_cap = "Add Query: ";
                   //echo "exask";exit;
                   for($i=0;$i<10;$i++){
                      $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
                      VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."',
'mongodb_cassandra_redis', '".date("Y-m-d H:i:s")."')");



                      $cusid = (int) mysql_insert_id();

                      //NEXT INSERT INTO Cassandra
                      $cluster   = Cassandra::cluster()->build();
                      $keyspace  = 'store';//database name
                      $session   = $cluster->connect($keyspace);
                      $session->execute(new Cassandra\SimpleStatement(
                            "INSERT INTO table_store (id, caption) VALUES
(now(),'".$stores[rand(0,10)]."')"
                         )
                      );

                      //NEXT INSERT INTO Redis
                      $redis = new Predis\Client();
                      $recrd = array(
                         'customers' => $customers[rand(0,10)],
                         'stores' => $stores[rand(0,10)],
                         'products' => $products[rand(0,10)],
                         'amount' => rand(500,10000),
                         'addedby' => $admins[rand(0,2)],
                         'addedon' => date("Y-m-d H:i:s"),

                         'updatefield' => 0
                      );
                      $redis->hmset('product'.$i, $recrd);

                      //NEXT INSERT INTO MongoDB
                      if($insertcat){
                         for($e=0;$e<200;$e++){
                            $bulk = new MongoDB\Driver\BulkWrite;
```

```
                    $document = [
                                'id' => new MongoDB\BSON\ObjectId,
                                'customerid' => $cusid,
                                'customers' => $customers[rand(0,10)],
                                'store' => $stores[rand(0,10)],
                                'product' => $products[rand(0,10)],
                                'amount' => rand(500,10000),
                                'addedby' => $admins[rand(0,2)],
                                'updatefield' => 1,
                                'createdon' => date("Y-m-d H:i:s")
                    ];
                    $bulk->insert($document);
                    $mng->executeBulkWrite('products.records', $bulk);
                 }
              }
           }



           $endtime= microtime(true);
        }elseif($opt=="edit"){
           $display_cap = "Update Query: ";
           for($i=0;$i<$limit;$i++){
              mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
           }
           $endtime= microtime(true);
           //RESET EVERYTHING AGAIN,
           mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
        }elseif($opt=="delete"){
           $display_cap = "Delete Query: ";
           mysql_query("DELETE FROM sales");
           $endtime= microtime(true);
        }else{//for read timer
           $display_cap = "Read Query: ";
        }
     ?>

   <!-- /.box-header -->
   <div class="box-body table-responsive no-padding">
     <table class="table table-hover">
       <tr>
         <th>#</th>
         <th>Customers</th>
         <th>Gender</th>
         <th>Phone</th>
         <th>Email</th>
         <th>Added On</th>
         <th>Action</th>
       </tr>

       <?php
```

```
          connectAppDB();
          $myquery = "SELECT * FROM users WHERE salestype = 'mongodb_cassandra_redis'
ORDER BY id DESC";
          $list = mysql_query($myquery);
          $total_rows = mysql_num_rows($list);
          //exit;
          if($total_rows > 0){
            $i=1;
            while($record = mysql_fetch_array($list)){
        ?>
            <tr>
              <td><?=$i?></td>
              <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
              <td><?=$record['gender']?></td>



              <td><?=$record['phone']?></td>
              <td><?=$record['email']?></td>
              <td><?=system_date($record['createon'],2)?></td>
              <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
              <td>
                <a class="label label-info"
                onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
              </td>
            </tr>
        <?php
              $i++;
            }
          }
          if($opt==""){//for read
            $endtime= microtime(true);
          }
          $timediff = $endtime - $starttime;
          $time_elapsed = secondsToTime($timediff);
        ?>
       </table>
      </div>
      <!-- /.box-body -->
     </div>
     <!-- /.box -->
    </div>
   </div>
  </section>
  <!-- /.content -->
 </div>


 <!--MODAL FORM FOR ADDING RECORDS TO MONGODB DATABASE -->
  <form name="new_record" action="controller.php" method="POST">
```

```
        <div class="modal fade" id="modal-default5">
          <div class="modal-dialog">
            <div class="modal-content">
              <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                  <span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From
Cassandra/MongoDB NoSQl Database</h4>
              </div>
              <div class="modal-body">

                <div class="box-body">




                    <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
                    <div id="show_display"></div>
                </div>

              </div>
              <div class="modal-footer">
                <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
              </div>
            </div>
            <!-- /.modal-content -->
          </div>
          <!-- /.modal-dialog -->
        </div>
    </form>

    <script type="text/javascript">
        $(document).ready(function() {
            $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
            $('#record_rows').html("[Total Records: <?=$total_rows?>]");
        });    //end of document ready function
        function askDelete(){
            if(confirm("Delete All Records?")){
                parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
            }
        }
        function callModalSales(ith){
            $("#modal-default5").modal("show");
            $("#customerr").html($("#customer"+ith).html());
            $("#show_display").html("");
            $('#txtHint').show();
            var loadUrl = "controller.php?action=view_mongo_sales&customerid="+ith;
            $.ajax({
                type: "POST",
                url: loadUrl,
                cache: false,
                success: function(output){
                    $("#show_display").html(output);
                    $('#txtHint').hide();
                },
                error: function(xhr, textStatus, errorThrown) {
                    $('#txtHint').hide();
                }
            });
        }



    </script>
<?    getPageFooter()?>
```

MYSQL_CASSANDRA_REDIS_NEO4J_MONGODB

```
<?php
require_once("functions.php");
require_once("pagelayout.php");

require "predis/autoload.php";
Predis\Autoloader::register();


require_once 'neo4j/graphaware/vendor/autoload.php';
use GraphAware\Neo4j\Client\ClientBuilder,
    GraphAware\Neo4j\OGM\EntityManager;




$client = ClientBuilder::create()
    ->addConnection('default', 'http://neo4j:userpass@localhost:7474') // Example for HTTP connection
configuration (port is optional)
    ->addConnection('bolt', 'bolt://neo4j:userpass@localhost:7687') // Example for BOLT connection
configuration (port is optional)
    ->build();




getAppHeader("MySQL - Cassandra - Redis -Neo4j - MongoDB");
getCloseHeader();
getHeaderTag();
getSideBar();

$time_elapsed = "0";
$opt = "";
$display_cap = "";
if(isset($_GET['opt'])){
    $opt = $_GET['opt'];
```

```
}

   $limit = 50;
?>




  <!-- Content Wrapper. Contains page content -->
  <div class="content-wrapper">
    <!-- Content Header (Page header) -->
    <section class="content-header">
      <h1>
        MySQL - Cassandra - Redis - Neo4j - MongoDB System
        <span id="display_results" class="btn-success"></span>
        <small style="float:right"><a href="./" title="click to go back"><b><< Go Back</b></a></small>
      </h1>

    </section>

    <!-- Main content -->
    <section class="content">

    <!-- /.row -->

      <? callErrorMessage();?>

      <div class="row">
        <div class="col-xs-12">
          <div class="box">

            <div class="box-header">
              <h3 class="box-title">Customer Records Directly From MYSQL (RDBMS)</h3>

              <span class="btn-info" id="record_rows"></span>

              <div class="box-tools">

                <a href="<?=$_SERVER['SCRIPT_NAME']?>" class="btn btn-info">Read All Records</a>
                <a href="<?=$_SERVER['SCRIPT_NAME']."?opt=add"?>" class="btn btn-success">Add
Records</a>
                <!--<a style="cursor:pointer" onclick="askDelete()" class="btn btn-danger">Delete
Records</a>-->
              </div>
            </div>

            <?php

                    //DATA FOR MONGODB, SALES INFORMATION
                    $customers =
array("John","Adam","Ugonna","Bayo","Sheyi","Miracle","Emeka","Segun","Udoka","Chibuzo","Akani
mo");
                    $stores = array("Store 1","Store 2","Store 3","Store 4","Store 5","Store 6","Store


7","Store 8","Store 9","Store 10","Store 11");
                    $products = array("Product 1","Product 2","Product 3","Product 4","Product 5","Product
6","Product 7","Product 8","Product 9","Product 10","Product 11");
                    $admins = array("Ofoefule Christian","Ekene Okeke","Ogundare Ayo");
                    $category = array("Electronics","Bags","Shoes","Shirts","Beverages");
                    // start script execution timer here

                    //DATA FOR MYSQL, USER INFORMATION
                    $gender = array("Male","Female");
                    $phone =
array("08011221123","08011221122","08011221123","08011220022","08011341123","08010921122","
08011871123","08011551122");
                    $email =
array("user12@gmail.com","user13@gmail.com","user14@gmail.com","user15@gmail.com","user16@g
mail.com","user17@gmail.com","user18@gmail.com","user19@gmail.com");

                    // start script execution timer here
                    $starttime = microtime(true);

                    //CONNECT TO MYSQL
                    connectAppDB();

                    //CONNECT TO MONGODB
                    $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");
                    $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
                    $res = $mng->executeCommand("products", $stats);
                    $stats = current($res->toArray());

                    if($opt=="add"){//add records
                       $display_cap = "Add Query: ";
                       //echo "exask";exit;
                       for($i=0;$i<10;$i++){
                          $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone,
salestype, createon)
                          VALUES

('".$customers[rand(0,10)]."','".$gender[rand(0,1)]."','".$email[rand(0,7)]."','".$phone[rand(0,7)]."',
'mongodb_cassandra_redis_neo4j', '".date("Y-m-d H:i:s")."')");
                          $cusid = (int) mysql_insert_id();

                          //NEXT INSERT INTO Cassandra
                          $cluster   = Cassandra::cluster()->build();
                          $keyspace  = 'store';//database name
                          $session   = $cluster->connect($keyspace);
                          $session->execute(new Cassandra\SimpleStatement(
                                "INSERT INTO table_store (id, caption) VALUES
(now(),'".$stores[rand(0,10)]."')"
                    )



                          );

                          //NEXT INSERT INTO Redis
                          $redis = new Predis\Client();
                          $recrd = array(
                    'customers' => $customers[rand(0,10)],
                    'stores' => $stores[rand(0,10)],
                    'products' => $products[rand(0,10)],
                    'amount' => rand(500,10000),
                    'addedby' => $admins[rand(0,2)],
                    'addedon' => date("Y-m-d H:i:s"),
                    'updatefield' => 0
                          );
                          $redis->hmset('product'.$i, $recrd);

                          //NEXT INSERT INTO Neo4j
                          $client->run('CREATE (n:Product_Category) SET n += {infos}',
                    ['infos' =>
                                [
                                    'caption' => $category[rand(0,4)]
                                ]
                    ]
                          );

                          //NEXT INSERT INTO MongoDB
                          if($insertcat){
                    for($e=0;$e<200;$e++){
                                $bulk = new MongoDB\Driver\BulkWrite;
                                $document = [
                                        'id' => new MongoDB\BSON\ObjectId,
                                        'customerid' => $cusid,
                                        'customers' => $customers[rand(0,10)],
                                        'store' => $stores[rand(0,10)],
                                        'product' => $products[rand(0,10)],
                                        'amount' => rand(500,10000),
                                        'addedby' => $admins[rand(0,2)],
                                        'updatefield' => 1,
                                        'createdon' => date("Y-m-d H:i:s")
                                ];
                                $bulk->insert($document);
                                $mng->executeBulkWrite('products.records', $bulk);
                    }
                          }
                       }
                       $endtime= microtime(true);
                   }elseif($opt=="edit"){
                       $display_cap = "Update Query: ";



                       for($i=0;$i<$limit;$i++){
                          mysql_query("UPDATE sales SET updatefield = 1 WHERE updatefield = 0 LIMIT 1");
                       }
                       $endtime= microtime(true);
                       //RESET EVERYTHING AGAIN,
                       mysql_query("UPDATE sales SET updatefield = 0 WHERE updatefield = 1");
                   }elseif($opt=="delete"){
                       $display_cap = "Delete Query: ";
                       mysql_query("DELETE FROM sales");
                       $endtime= microtime(true);
                   }else{//for read timer
                       $display_cap = "Read Query: ";
                   }
               ?>

             <!-- /.box-header -->
             <div class="box-body table-responsive no-padding">
               <table class="table table-hover">
                 <tr>
                   <th>#</th>
                   <th>Customers</th>
                   <th>Gender</th>
                   <th>Phone</th>
                   <th>Email</th>
                   <th>Added On</th>
                   <th>Action</th>
                 </tr>

                 <?php
                   connectAppDB();
                   $myquery = "SELECT * FROM users WHERE salestype =
       'mongodb_cassandra_redis_neo4j' ORDER BY id DESC";
                   $list = mysql_query($myquery);
                   $total_rows = mysql_num_rows($list);
                   //exit;
                   if($total_rows > 0){
                       $i=1;
                       while($record = mysql_fetch_array($list)){
                 ?>
                       <tr>
                          <td><?=$i?></td>
                          <td id="customer<?=$record['id']?>"><?=$record['usernames']?></td>
                          <td><?=$record['gender']?></td>
                          <td><?=$record['phone']?></td>
                          <td><?=$record['email']?></td>
                          <td><?=system_date($record['createon'],2)?></td>
```

```
                        <span id="customerid<?=$record['id']?>"
style="display:none"><?=$record['id']?></span>
                        <td>
                            <a class="label label-info"
                            onclick="callModalSales('<?=$record['id']?>')">View All Purchases</a>
                        </td>
                    </tr>
                <?php
                        $i++;
                    }
                }
                if($opt==""){//for read
                    $endtime= microtime(true);
                }
                $timediff = $endtime - $starttime;
                $time_elapsed = secondsToTime($timediff);
            ?>
            </table>
          </div>
          <!-- /.box-body -->
        </div>
        <!-- /.box -->
      </div>
    </div>
  </section>
  <!-- /.content -->
 </div>


 <!--MODAL FORM FOR ADDING RECORDS TO MONGODB DATABASE -->
  <form name="new_record" action="controller.php" method="POST">
      <div class="modal fade" id="modal-default5">
        <div class="modal-dialog">
          <div class="modal-content">
            <div class="modal-header">
              <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                <span aria-hidden="true">&times;</span></button>
              <h4 class="modal-title"><span id="customerr"></span>'s Sales Records From
Cassandra/MongoDB NoSQl Database</h4>
            </div>
            <div class="modal-body">

              <div class="box-body">

                  <div class="uk-text-center" style="display:none" id="txtHint"><img
src="img/spinner.gif"></div>
```

```
                <div id="show_display"></div>
              </div>

            </div>
            <div class="modal-footer">
              <button type="button" class="btn btn-default pull-left" data-dismiss="modal">Close</button>
            </div>
          </div>
          <!-- /.modal-content -->
        </div>
        <!-- /.modal-dialog -->
      </div>
    </form>

    <script type="text/javascript">
      $(document).ready(function() {
        $('#display_results').html("[<?=$display_cap.$time_elapsed?>]");
        $('#record_rows').html("[Total Records: <?=$total_rows?>]");
      });    //end of document ready function
      function askDelete(){
        if(confirm("Delete All Records?")){
          parent.location="<?=$_SERVER['SCRIPT_NAME']."?opt=delete"?>";
        }
      }
      function callModalSales(ith){
        $("#modal-default5").modal("show");
        $("#customerr").html($("#customer"+ith).html());
        $("#show_display").html("");
        $('#txtHint').show();
        var loadUrl = "controller.php?action=view_mongo_sales&customerid="+ith;
        $.ajax({
          type: "POST",
          url: loadUrl,
          cache: false,
          success: function(output){
            $("#show_display").html(output);
            $('#txtHint').hide();
          },
          error: function(xhr, textStatus, errorThrown) {
            $('#txtHint').hide();
          }
        });
      }
    </script>
<?    getPageFooter()?>
```

FUNCTIONS.PHP

```
<?php

session_start();
date_default_timezone_set('Africa/Lagos');

function connectAppDB() {
   $dbhost = 'localhost';
   $dbuser = 'root';//christhi_user01
   $dbpassword = 'server.cloud';//christ.hill.pass01
   $connect = @mysql_connect($dbhost, $dbuser, $dbpassword);//using @ to suppress any
possible error that may arise from using this function
   if(!$connect) {
      die('Connection failed: ' . mysql_error());
      return false;
   }
   //select a apecific DB/Schema
   $dbname = 'ekene';
   mysql_select_db($dbname,$connect);
   if(!mysql_select_db($dbname)) {
      die('Selected database unavailable: ' . mysql_error());
      return false;
   }
   return $connect;
}
function getCleanString($input){
   $clean = mysql_real_escape_string(trim($input));
   return $clean;
}
function getDateTime() {
   $format = "Y-m-d H:i:s";
   $today =  date($format);
   return $today;
}
function secondsToTime2($s){
   $h = floor($s / 3600);
   $s -= $h * 3600;
   $m = floor($s / 60);
   $s -= $m * 60;
   return $h.':'.sprintf('%02d', $m).':'.sprintf('%02d', $s);
}
function secondsToTime($s){
   return round($s,5)." secs";
}
```

```
function system_date($date,$mode=1){
    if(trim($date)!=""){
        if($mode==1){
            return date("jS-M-Y", strtotime($date));
        }else{
            return date("jS-M-Y h:i:s A", strtotime($date));
        }
    }
}
function format_number($number,$default=2){
    if($default==2){
        return number_format($number,2);
    }else{
        return number_format($number);
    }
}
?>
```

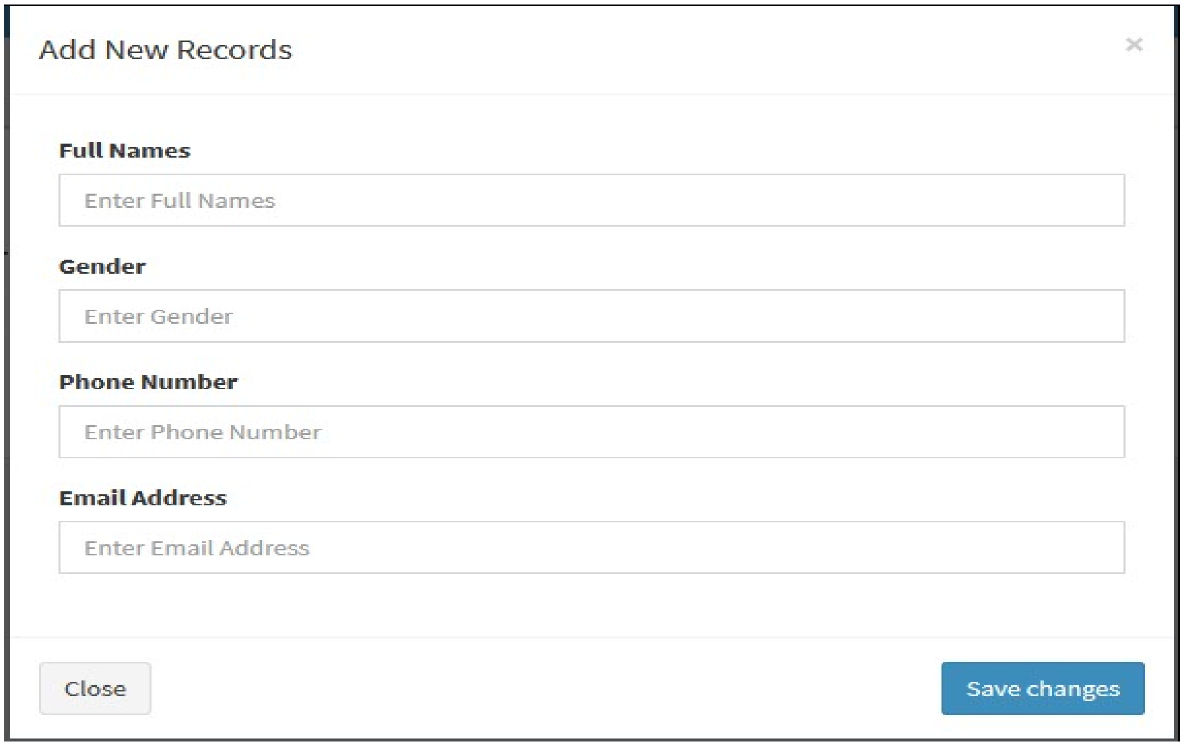
CONTROLLER.PHP

```
<?php
require_once("functions.php");

//use commands can only be on top of page script
require_once 'neo4j/graphaware/vendor/autoload.php';
use GraphAware\Neo4j\Client\ClientBuilder,
        GraphAware\Neo4j\OGM\EntityManager;


if(isset($_REQUEST['action'])) {
    $user_action = $_REQUEST['action'];
    $myresponse = "login";
    $_SESSION['error'] = array();

    if($user_action == 'new_record')
        $myresponse = newRecord();
      else if($user_action == 'new_record_store')
        $myresponse = newRecordStore();
      else if($user_action == 'new_record_category')
        $myresponse = newRecordCategory();
    else if($user_action == 'new_record_sale')
        $myresponse = newRecordSale();
    else if($user_action == 'view_mongo_sales')
        $myresponse = mongoDBSales();
    else if($user_action == 'view_cassandra_sales')
        $myresponse = cassandraSales();
    else if($user_action == 'view_redis_sales')
        $myresponse = redisSales();
    else if($user_action == 'view_neo4j_sales'){
        $myresponse = neo4jSales();
    }







   header('Location: ' . $myresponse);//
    exit;
}
function newRecord(){
    try{
        $con = connectAppDB();// COnnect to Database
        $user = getCleanString($_REQUEST['usernames']);
        $gender = getCleanString($_REQUEST['gender']);
        $phone = getCleanString($_REQUEST['phone']);
        $email = getCleanString($_REQUEST['email']);
        //First of all, the 2 passwords must match

        $insertcat = mysql_query("INSERT INTO users (usernames, gender, email, phone, createon)
        VALUES
        ('".$user."','".$gender."','".$email."','".$phone."','".date("Y-m-d H:i:s")."')");

        if ($insertcat != ""){
            $id = (int) mysql_insert_id();
        $_SESSION['divborder'] = "success";
        $_SESSION['error'][] = 'New Customer Record Added to MySQL successfully';

        }else{
            $_SESSION['error'][] = 'Sorry, an error occurred. Please try again later.';
        }

        $nextURL = "index.php";
    }catch(Exception $ex){
        $_SESSION['error'][] = $ex->getMessage();
    }
    return $nextURL;
}
function newRecordStore(){

    $caption = $_REQUEST['caption'];

    $cluster   = Cassandra::cluster()->build();
    $keyspace  = 'store';//database name
    $session   = $cluster->connect($keyspace);        // create session, optionally scoped to a keyspace

    //QUERY TO CREATE A DATABASE
    //CREATE KEYSPACE store WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 2 };

    //QUERY TO SELEC A DATABASE
    //USE MyKeySpace;

    //QUERY TO CREATE A TABLE
    //CREATE COLUMNFAMILY table_store (id uuid, Caption text, PRIMARY KEY(id));
    //QUERY TO SELECT RECORDS FROM TABLE
    //SELECT * FROM table

    //QUERY TO DROP TABLE
    //DROP TABLE IF EXISTS store.table_store







    //QUERY TO ADD NEW RECORD TO STORE TABLE
    $statement = new Cassandra\SimpleStatement("INSERT INTO table_store (id, caption) VALUES
(now(),'".$caption."')");

    $future    = $session->executeAsync($statement);  // fully asynchronous and easy parallel execution
    $result    = $future->get();                      // wait for the result, with an optional timeout

    $_SESSION['divborder'] = "success";
    $_SESSION['error'][] = 'New Store Record Added to Cassandra Database Successfully';

    $nextURL = "index.php";
    return $nextURL;
    /*
    foreach ($result as $row) {
        printf("The keyspace %s has a table called %s\n", $row['keyspace_name'], $row['columnfamily_name']);
        echo "<br>";
    }
    */
}
function newRecordCategory(){

    require "../predis/autoload.php";
    Predis\Autoloader::register();

    //The number of Redis databases is fixed, and set in the configuration file. By default, you have 16 databases

    try {
        $caption = $_REQUEST['caption'];

        $redis = new Predis\Client();

        //$redis->set('message', 'Hello world');

        // gets the value of message
        //$value = $redis->get('message');

        // Hello world
        //print($value);

        //echo ($redis->exists('message')) ? "Oui" : "please populate the message key";

        $user_id = $redis->incr('table_category:id');
        $redis->hmset('table_category:' . $user_id, 'caption', $caption);
        $redis->set('table_category:caption:'.$caption, $user_id);
        $redis->zadd('table_category:created_at', time(), $user_id);

        $_SESSION['divborder'] = "success";
        $_SESSION['error'][] = 'New Store Record Added to Redis Database Successfully';

    }catch (Exception $e) {
        die($e->getMessage());
    }

    $nextURL = "index.php";
    return $nextURL;





}
function newRecordSale(){
    try {
        $mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");
        $stats = new MongoDB\Driver\Command(["dbstats" => 1]);
        $res = $mng->executeCommand("products", $stats);
        $stats = current($res->toArray());
        $bulk = new MongoDB\Driver\BulkWrite;

        $customerid = $_REQUEST['customerid'];
        $storeid = $_REQUEST['storeid'];
        $productid = $_REQUEST['productid'];
        $amount = $_REQUEST['amount'];
        $document = [
                        'id' => new MongoDB\BSON\ObjectId,
                        'customerid' => $customerid,
                        'store' => $storeid,
                        'product' => $productid,
                        'amount' => $amount,
                        'createdon' => date("Y-m-d H:i:s")
        ];

        $bulk->insert($document);
        $mng->executeBulkWrite('products.records', $bulk);
        $_SESSION['divborder'] = "success";
        $_SESSION['error'][] = 'New Sale Record Added to MongoDB Database Successfully';

    } catch (MongoDB\Driver\Exception\Exception $e) {

        $filename = basename(__FILE__);

        echo "The $filename script has experienced an error.\n";
        echo "It failed with the following exception:\n";

        echo "Exception:", $e->getMessage(), "\n";
        echo "In file:", $e->getFile(), "\n";
        echo "On line:", $e->getLine(), "\n";
        exit;
    }
    $nextURL = "index.php";
    return $nextURL;
}
function neo4jSales(){
    $customerid = $_REQUEST['customerid'];
    $starttime = microtime(true);
    $client = ClientBuilder::create()
        ->addConnection('default', 'http://neo4j:userpass@localhost:7474') // Example for HTTP connection
configuration (port is optional)
        ->addConnection('bolt', 'bolt://neo4j:userpass@localhost:7687') // Example for BOLT connection configuration
(port is optional)
        ->build();
    $result = $client->run('MATCH (n:Sales) WHERE n.customerid='.($customerid).' RETURN n');
    $total_rows=0;
    echo '<span id="sales_rows" class="btn-success"></span><br><br>';





    echo "
            <table class='table table-hover'>
                <tr>
                    <th>#</th>
                    <th>Store/Branch</th>
                    <th>Product</th>
                    <th>Amount(&#8358;)</th>
                    <th>Added By</th>
                    <th>Date</th>
                </tr>
    ";
    $n=1;
    $total_rows=0;
    foreach ($result->getRecords() as $record) {
        $total_rows++;
        echo "
            <tr>
                <td>".$n."</td>
                <td>".$record->get('n')->value('stores')."</td>
                <td>".$record->get('n')->value('products')."</td>
                <td>".format_number($record->get('n')->value('amount'))."</td>
                <td>".$record->get('n')->value('addedby')."</td>
                <td>".system_date($record->get('n')->value('addedon'),2)."</td>
            </tr>
        ";
        $n++;
    }
    $endtime= microtime(true);
    $timediff = $endtime - $starttime;
    $time_elapsed = secondsToTime($timediff);
    echo '
            <script type="text/javascript">
                $(document).ready(function() {
                    //$("#display_results").html("[");
                    $("#sales_rows").html("[Total Records: '.$total_rows.']; [Total Time: '.$time_elapsed.']");
                });
            </script>
    ';
    exit;
}
function redisSales(){
    require "predis/autoload.php";
    $customerid = $_REQUEST['customerid'];
    $starttime = microtime(true);
    Predis\Autoloader::register();
    $total_rows=0;
    echo '<span id="sales_rows" class="btn-success"></span><br><br>';
    echo "
            <table class='table table-hover'>
                <tr>
                    <th>#</th>
                    <th>Store/Branch</th>
                    <th>Product</th>
                    <th>Amount(&#8358;)</th>
                    <th>Added By</th>
                    <th>Date</th>
                </tr>



    ";
    $redis = new Predis\Client();
    $n=1;
    $total_rows=0;
    for($i=0;$i<200;$i++){
        $record = $redis->hgetall('record'.$customerid);
        $total_rows++;
        if(isset($record['customers'])){
            echo "
                <tr>
                    <td>".$n."</td>
                    <td>".$record['stores']."</td>
                    <td>".$record['products']."</td>
                    <td>".format_number($record['amount'])."</td>
                    <td>".$record['addedby']."</td>
                    <td>".system_date($record['addedon'],2)."</td>
                </tr>
            ";
            $n++;
        }
    }
    $endtime= microtime(true);
    $timediff = $endtime - $starttime;
    $time_elapsed = secondsToTime($timediff);
    echo '
            <script type="text/javascript">
                $(document).ready(function() {
                    //$("#display_results").html("[");
                    $("#sales_rows").html("[Total Records: '.$total_rows.']; [Total Time: '.$time_elapsed.']");
                });
            </script>
    ';
    exit;
```

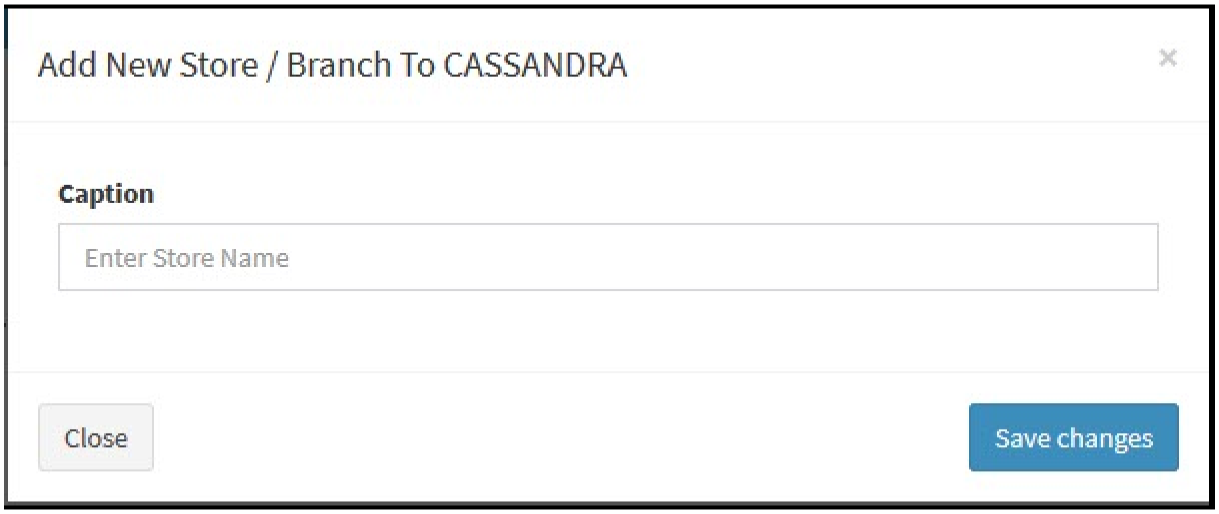
```
}
function cassandraSales(){
   $customerid = $_REQUEST['customerid'];
   $starttime = microtime(true);
   $cluster   = Cassandra::cluster()->build();
   $keyspace  = 'store';//database name
   $session   = $cluster->connect($keyspace);
   $total_rows=0;
   $result = $session->execute(new Cassandra\SimpleStatement(
   "SELECT * FROM sales WHERE customerid = '".$customerid."'"
   ));
   echo '<span id="sales_rows" class="btn-success"></span><br><br>';
   echo "
         <table class='table table-hover'>
            <tr>
               <th>#</th>
               <th>Store/Branch</th>
               <th>Product</th>
               <th>Amount(&#8358;)</th>
               <th>Added By</th>
               <th>Date</th>
            </tr>
   ";
   $total_rows=0;
   $n=1;
   foreach ($result as $record) {



      $total_rows++;
      echo "
         <tr>
            <td>".$n."</td>
            <td>".$record['stores']."</td>
            <td>".$record['products']."</td>
            <td>".format_number($record['amount'])."</td>
            <td>".$record['addedby']."</td>
            <td>".system_date($record['addedon'],2)."</td>
         </tr>
      ";
      $n++;
   }
   $endtime= microtime(true);
   $timediff = $endtime - $starttime;
   $time_elapsed = secondsToTime($timediff);
   echo '
         <script type="text/javascript">
            $(document).ready(function() {
               //$("#display_results").html("[");
               $("#sales_rows").html("[Total Records: '.$total_rows.']; [Total Time: '.$time_elapsed.']");
            });
         </script>
   ';
   exit;
}
function mongoDBSales(){
   $customerid = $_REQUEST['customerid'];
   $starttime = microtime(true);
   $mongo = new MongoDB\Driver\Manager("mongodb://localhost:27017");
   $filter      = ['customerid' => (int)$customerid];
   $options = [];
   $query = new \MongoDB\Driver\Query($filter, $options);
   $rows   = $mongo->executeQuery('products.records', $query);
   echo '<span id="sales_rows" class="btn-success"></span><br><br>';
   echo "
         <table class='table table-hover'>
            <tr>
               <th>#</th>
               <th>Store/Branch</th>
               <th>Product</th>
               <th>Amount(&#8358;)</th>
               <th>Added By</th>
               <th>Date</th>
            </tr>
   ";
   $d=1;
   $total_rows=0;
   foreach ($rows as $document) {
      $total_rows++;
      echo "
         <tr>
           <td>".$d."</td>
           <td>".$document->store."</td>
           <td>".$document->product."</td>
           <td>".format_number($document->amount)."</td>
           <td>".@$document->addedby."</td>
           <td>".system_date($document->createdon)."</td>



         </tr>
      ";
      $d++;
   }
   $endtime= microtime(true);
   $timediff = $endtime - $starttime;
   $time_elapsed = secondsToTime($timediff);
   echo '
         <script type="text/javascript">
            $(document).ready(function() {
               //$("#display_results").html("[");
               $("#sales_rows").html("[Total Records: '.$total_rows.']; [Total Time: '.$time_elapsed.']");
            });
         </script>
   ';
   exit;
}
?>
```

APPENDIX B

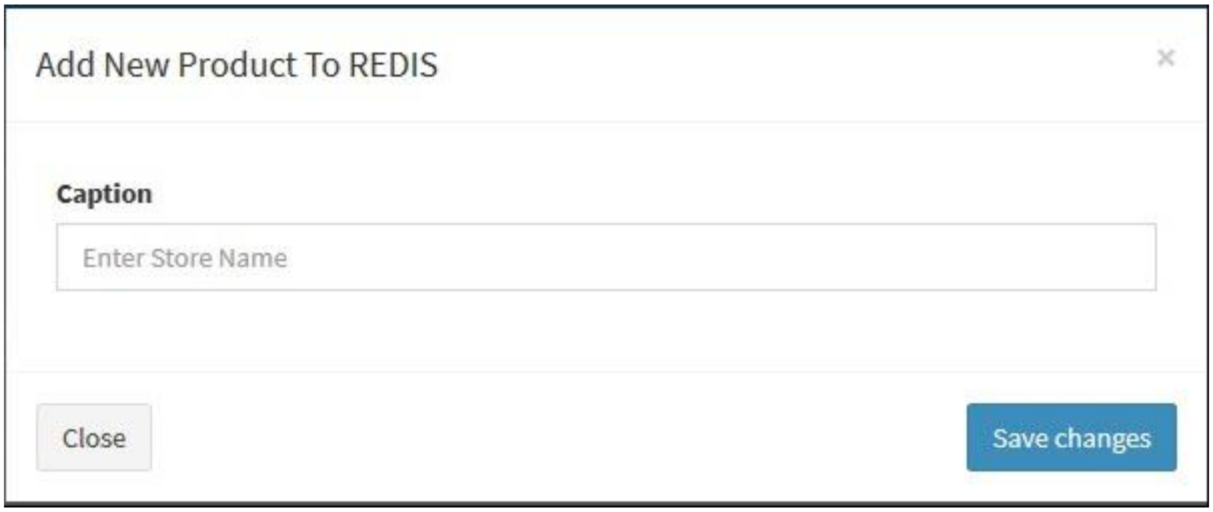
SAMPLE OUTPUTS



Form Interface to store customer information into MySQL database



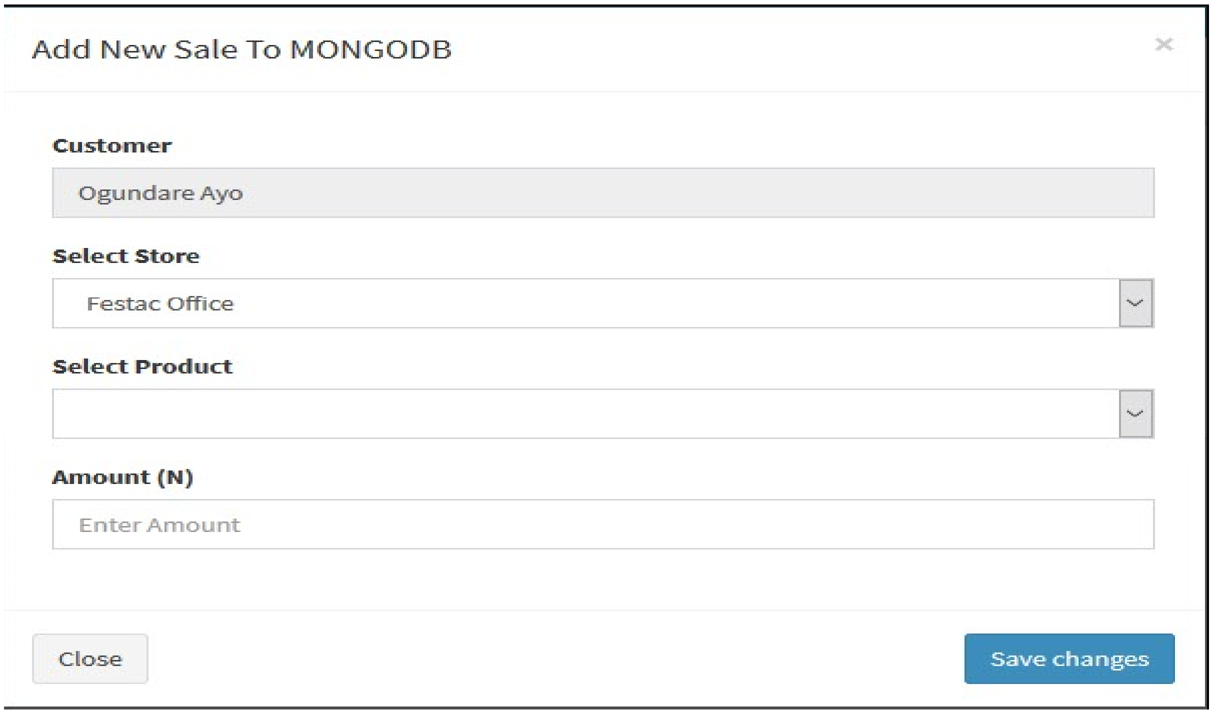
Form Interface to store branch data into Cassandra NoSQL


Add New Product To REDIS

Caption

Enter Store Name

Form Interface to store Product information to Redis


Add New Sale To MONGODB

Customer

Ogundare Ayo

Select Store

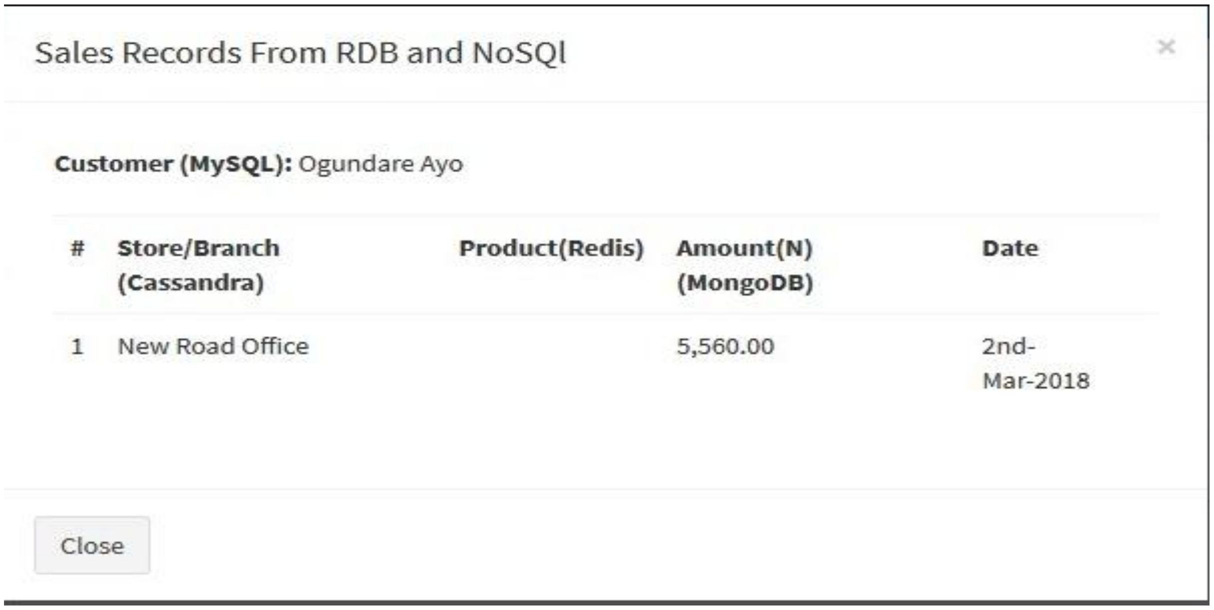
Festac Office

Select Product

Amount (N)

Enter Amount

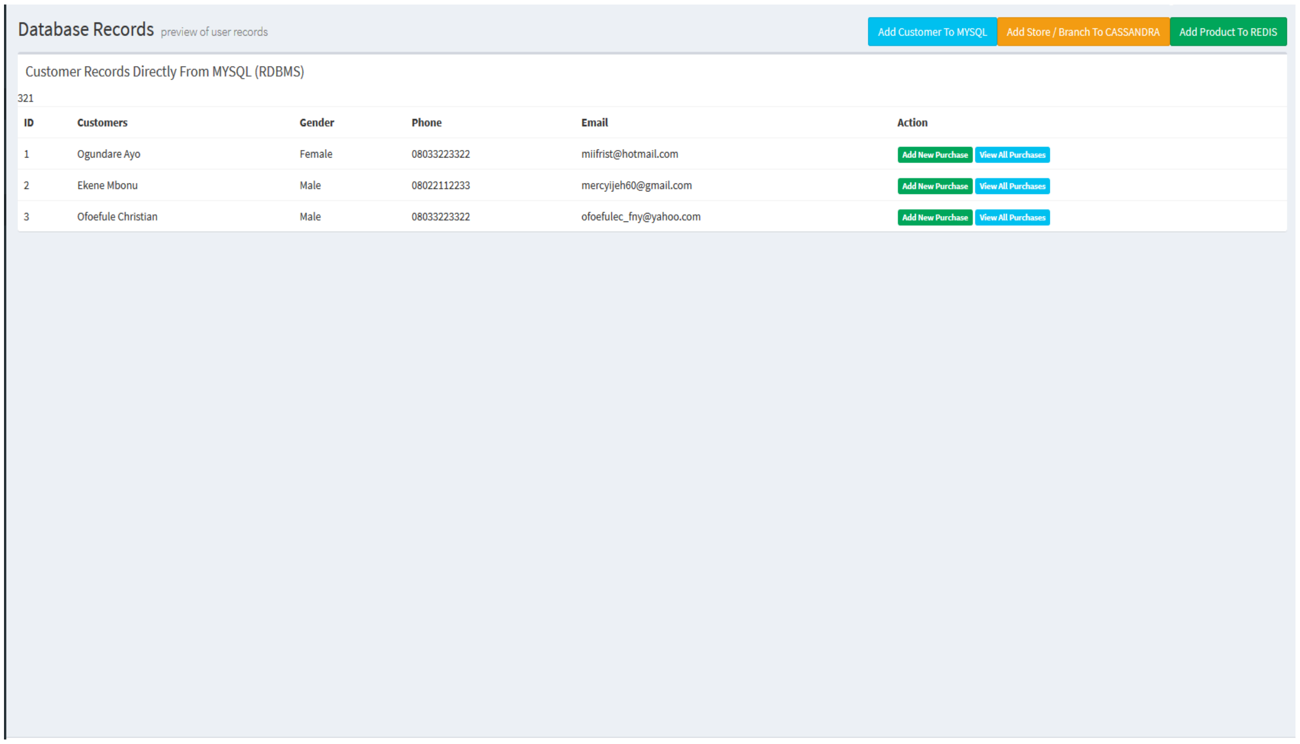
Form Interface to store New Sale Information to MongoDB


Sales Records From RDB and NoSQl

Customer (MySQL): Ogundare Ayo

#	Store/Branch (Cassandra)	Product(Redis)	Amount(N) (MongoDB)	Date
1	New Road Office		5,560.00	2nd-Mar-2018

Sales Information displaying records from Mysql (RDB) holding customer records, Cassandra Nosql holding branch records, MongoDB Nosql holding sales records, amount and date of sales and Redis NoSQL holding product information


Database Records preview of user records

Customer Records Directly From MYSQL (RDBMS)

321

ID	Customers	Gender	Phone	Email	Action
1	Ogundare Ayo	Female	08033223322	miifrist@hotmail.com	Add New Purchase View All Purchases
2	Ekene Mbonu	Male	08022112233	mercyijeh60@gmail.com	Add New Purchase View All Purchases
3	Ofoefule Christian	Male	08033223322	ofoefulec_fny@yahoo.com	Add New Purchase View All Purchases

Customer data information from MySQL (RDBMS). Graphical User Interface showing Action buttons

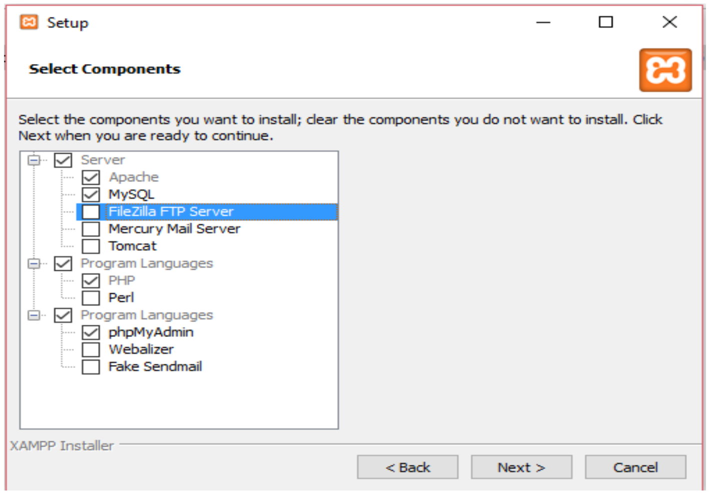
Limitations of the Initial Sample Outputs

The sample outputs displayed above did not have any means of testing for speed and performance as the records were manually added to the different databases which is not a true representation of the realtime depiction of the functionality of the systems involved and how they operate.

For the purpose of this downside of inability to automatically check for speed and movement of data and also for validation purposes, simulations needed to be conducted using massive records in order to ascertain the strengths and weaknesses of the various systems involved.

The initial tests conducted does not meet the requirements of the research as validation could not be carried out as manual input of data could not be used to test for latency and throughput which requires large volume of data to be sent to the database at the same time.

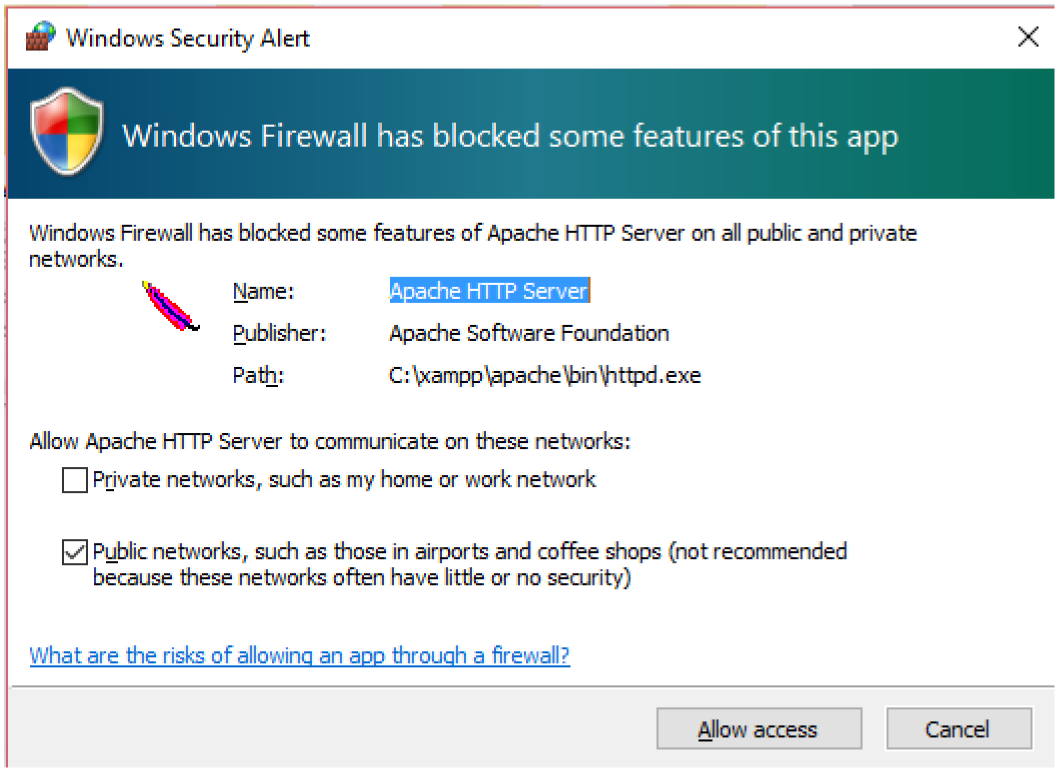
The researcher discarded this sytem and developed a more responsive system which was used to test for speed and performance of the various systems which also helped in deducing the strengths and limitations of the various systems (MySQL, MongoDB, Redis, Cassandra and Neo4j) involved.



Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Server
  - [✓] Apache
  - [✓] MySQL
  - [ ] FileZilla FTP Server
  - [ ] Mercury Mail Server
  - [ ] Tomcat
- [✓] Program Languages
  - [✓] PHP
  - [ ] Perl
- [✓] Program Languages
  - [✓] phpMyAdmin
  - [ ] Webalizer
  - [ ] Fake Sendmail

Installation of XAMPP Application: This application will install Apache Server, MySQL RDBM, and PHP software.


Windows Security Alert

Windows Firewall has blocked some features of this app

Windows Firewall has blocked some features of Apache HTTP Server on all public and private networks.

Name:	Apache HTTP Server
Publisher:	Apache Software Foundation
Path:	C:\xampp\apache\bin\httpd.exe

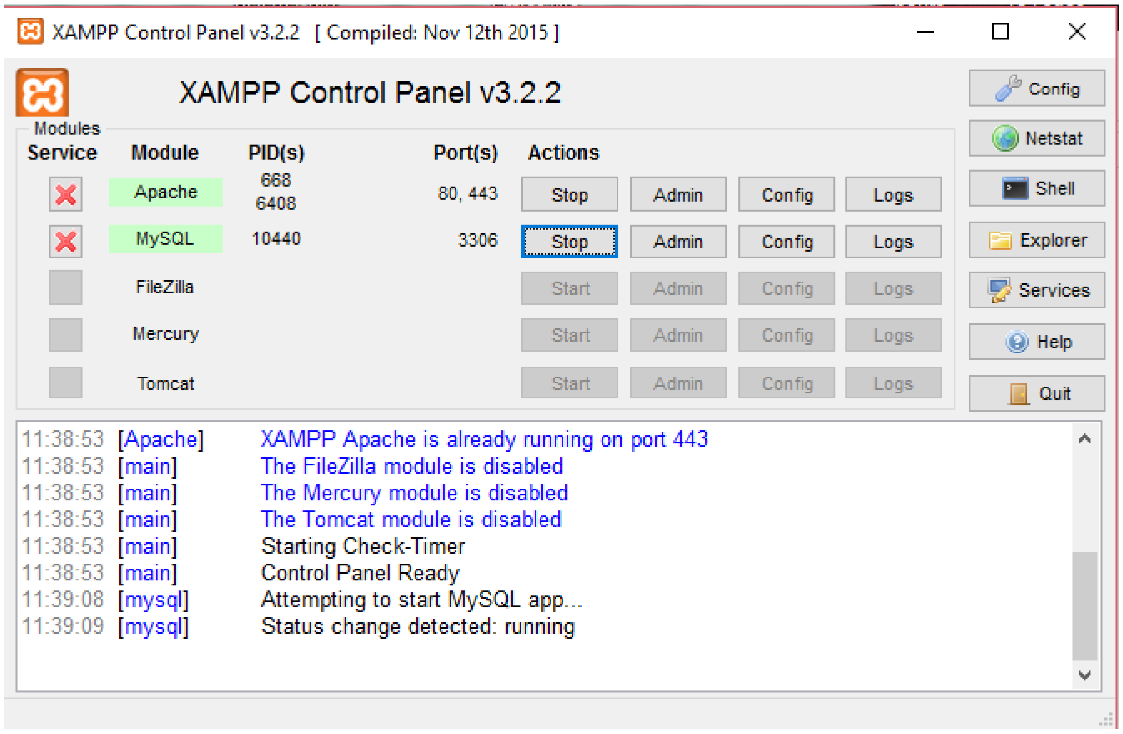
Allow Apache HTTP Server to communicate on these networks:

☐ Private networks, such as my home or work network

☑ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

Security Alert Encountered when Installing XAMPP Software

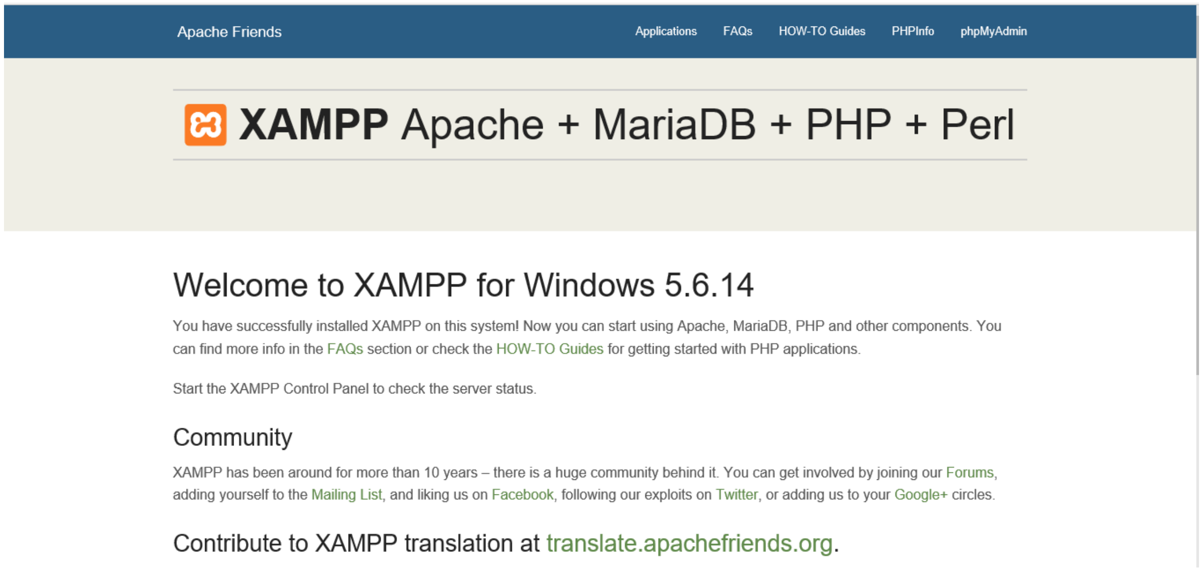

XAMPP Control Panel v3.2.2 [Compiled: Nov 12th 2015]

XAMPP Control Panel v3.2.2

Service	Module	PID(s)	Port(s)	Actions			
✗	Apache	668 6408	80, 443	Stop	Admin	Config	Logs
✗	MySQL	10440	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
11:38:53  [Apache]   XAMPP Apache is already running on port 443
11:38:53  [main]     The FileZilla module is disabled
11:38:53  [main]     The Mercury module is disabled
11:38:53  [main]     The Tomcat module is disabled
11:38:53  [main]     Starting Check-Timer
11:38:53  [main]     Control Panel Ready
11:39:08  [mysql]    Attempting to start MySQL app...
11:39:09  [mysql]    Status change detected: running
```

XAMPP Dashboard Control Panel



Welcome to XAMPP for Windows 5.6.14

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our Forums, adding yourself to the Mailing List, and liking us on Facebook, following our exploits on Twitter, or adding us to your Google+ circles.

Contribute to XAMPP translation at translate.apachefriends.org.

Complete Installation of XAMPP application Showing the Localhost Admin Page

Steps to Installing MongoDB

1. run installation file

2. open cmd prompt and navigate to

C:\Program Files\MongoDB\Server\3.6\bin

3. type mongodb to open power shell

4. now go to C drive and create these folders structure data/db

5. from the bin directory, run these commands

mongod.exe --dbpath c:\data

6. finally start the db using the command

start mongo.exe

MongoDB and PHP

1. install driver from

https://pecl.php.net/package/mongodb/1.2.8/windows i installed 5.6 Thread Safe (TS) x86

To Start a New Project after Installation

1. navigate to C:\Program Files\MongoDB\Server\3.6\bin

2. type mongod.

3. open a new cmd a

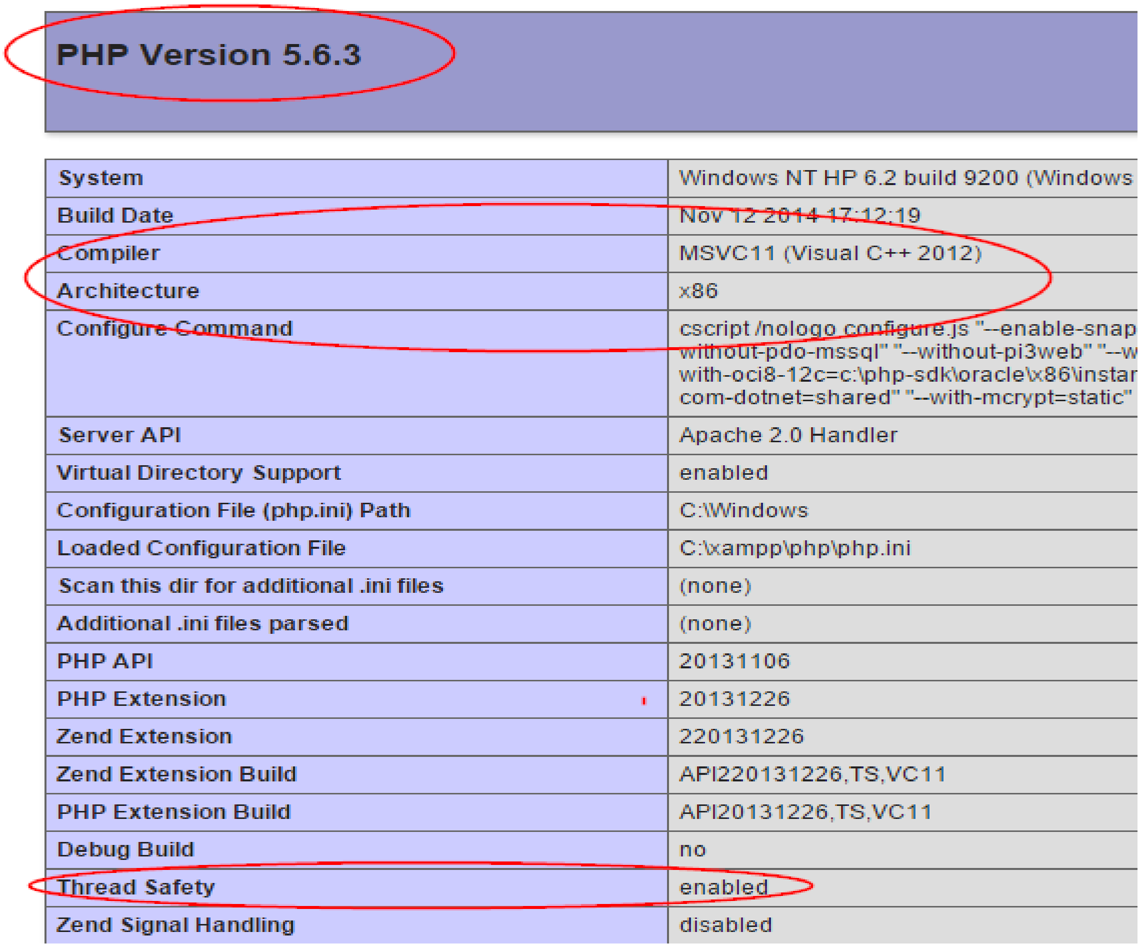
4. navigate to C:\Program Files\MongoDB\Server\3.6\bin

5. type mongo

Steps to Integration of PHP and MongoDB

1. Install and Configure XAMPP

First, you should install the XAMPP stack. Download and install the XAMPP stack from Apache friends project. Also, keep in mind that you can also install XAMPP on Linux as we discussed earlier. After the install, start your Apache server from XAMPP controls and creates a simple PHP file to get the detailed info about the PHP running with your stack. Just copy paste the below lines to a test.php file in the htdocs folder and execute it to see the output.



PHP Version 5.6.3

System	Windows NT HP 6.2 build 9200 (Windows
Build Date	Nov 12 2014 17:12:19
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snap without-pdo-mssql" "--without-pi3web" "--w with-oci8-12c=c:\php-sdk\oracle\x86\instar com-dotnet=shared" "--with-mcrypt=static"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,TS,VC11
PHP Extension Build	API20131226,TS,VC11
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled

2. Download PHP Mongo Driver

From this PHP Mongo Driver download page, download the appropriate file that matches the PHP version, Architecture, Compiler in use and Thread Safety from the XAMPP that is installed on your system.

3. Copy PHP Mongo DDL to EXT Directory

After you unzip the php monngo driver zip file, copy and paste the ".dll" file to the folder "C:\xampp\php\ext"(Assuming that xampp is installed on C drive).

This ext folder all the ".dll" files of all the extensions that are installed. XAMPP loads the driver files for the extensions from this folder. After copying the file over here, rename the ".dll" file to "php_mongo.dll" for simplicity.

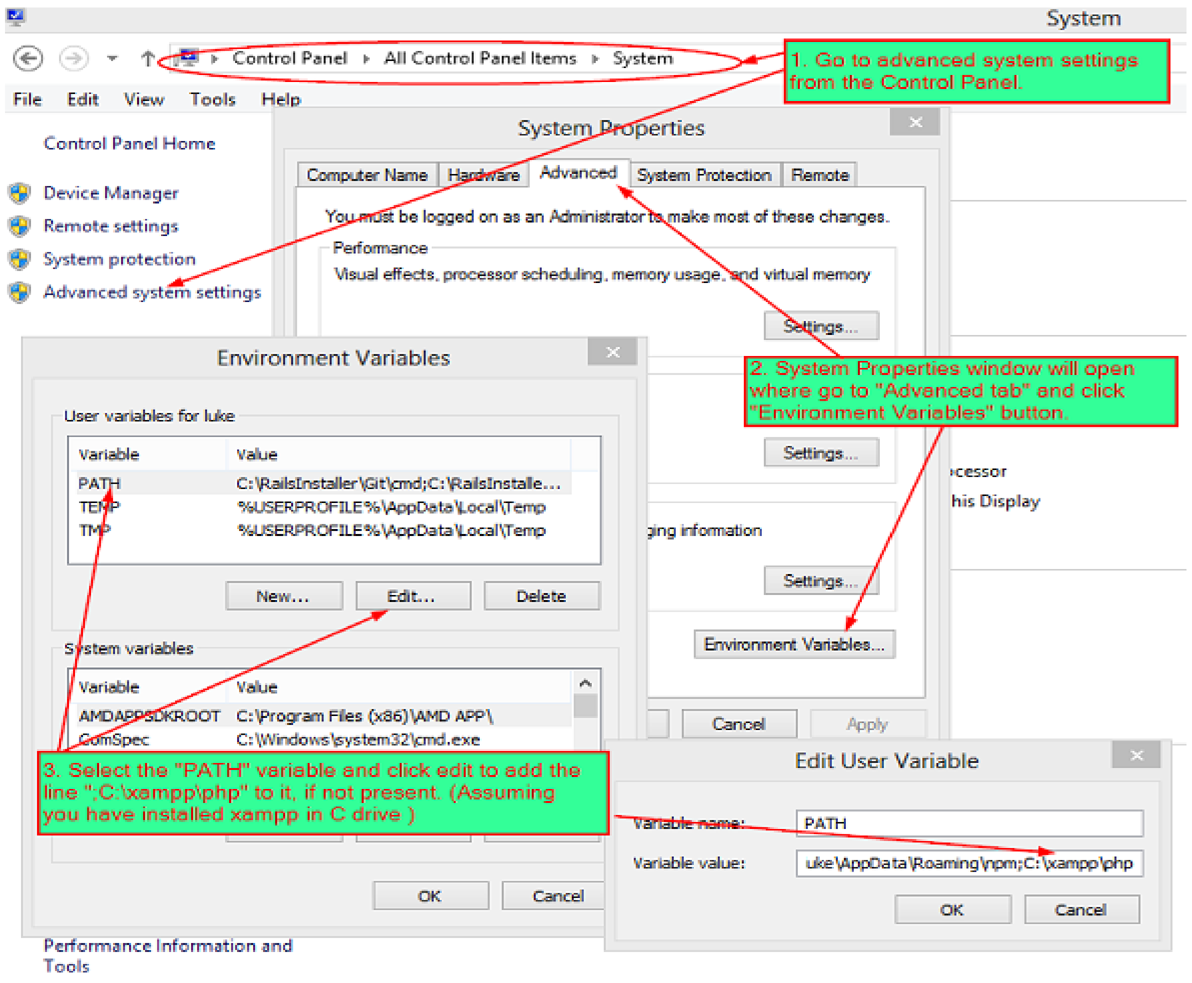
4. Add Extension to php.ini

Next, open the “php.ini” file from the path “C:\xampp\php” (again, assuming that xampp is installed on C drive), and edit this file to add the name of the “.dll” file as an extension. Add the following line to the php.ini file.

extension=php_mongo.dll

5. Modify the PATH Variable

Go to control panel, and open the system settings to add the “Environment Variable”. Add the path of the xampp php installation (C:\xampp\php) to the path variable, if it is not present already. This ensures that the newly added “.dll” file is loaded when xampp is started.



1. Go to advanced system settings from the Control Panel.

You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

2. System Properties window will open where go to "Advanced tab" and click "Environment Variables" button.

User variables for luke

Variable	Value
PATH	C:\RailsInstaller\Git\cmd;C:\RailsInstalle...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
AMDAPPSDKROOT	C:\Program Files (x86)\AMD APP\
ComSpec	C:\Windows\system32\cmd.exe

3. Select the "PATH" variable and click edit to add the line ";C:\xampp\php" to it, if not present. (Assuming you have installed xampp in C drive)

Variable name: PATH

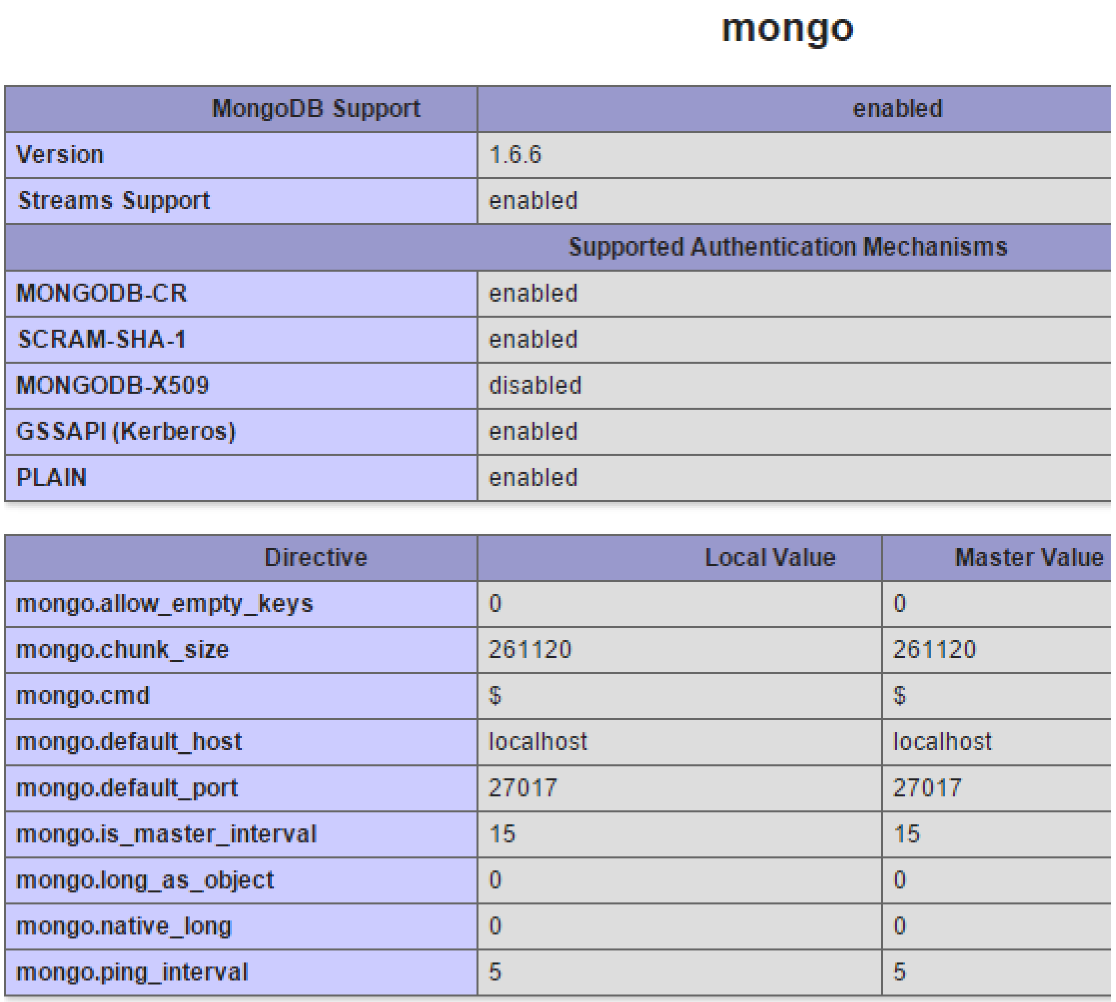
Variable value: uke\AppData\Roaming\npm;C:\xampp\php

6. Restart Apache and Verify

Finally, restart the Apache server from the XAMPP control panel.

If everything is configured properly, xampp should not throw any error messages while starting apache. You can also check the loaded extension by going through the first step and looking into the php information.

You will be able to see the loaded extension information on the page as shown below.



mongo

MongoDB Support	enabled
Version	1.6.6
Streams Support	enabled
Supported Authentication Mechanisms	
MONGODB-CR	enabled
SCRAM-SHA-1	enabled
MONGODB-X509	disabled
GSSAPI (Kerberos)	enabled
PLAIN	enabled

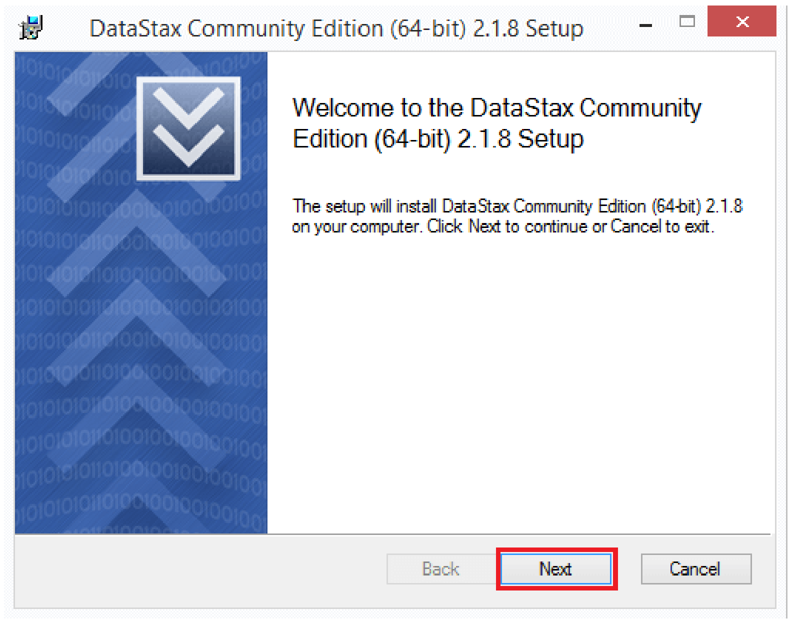
Directive	Local Value	Master Value
mongo.allow_empty_keys	0	0
mongo.chunk_size	261120	261120
mongo.cmd	$	$
mongo.default_host	localhost	localhost
mongo.default_port	27017	27017
mongo.is_master_interval	15	15
mongo.long_as_object	0	0
mongo.native_long	0	0
mongo.ping_interval	5	5

STEPS TO INSTALL CASSANDRA NO-SQL DATABASE

Apache Cassandra

Apache Cassandra is a free and open-source distributed NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure

Step 1) Run the Datastax community edition setup. After running the Setup, following page will be displayed. Here in the screenshot 64 bit version is being installed. You can download 32 bit version as well according to your requirements. But I recommend 64 bit version to use.

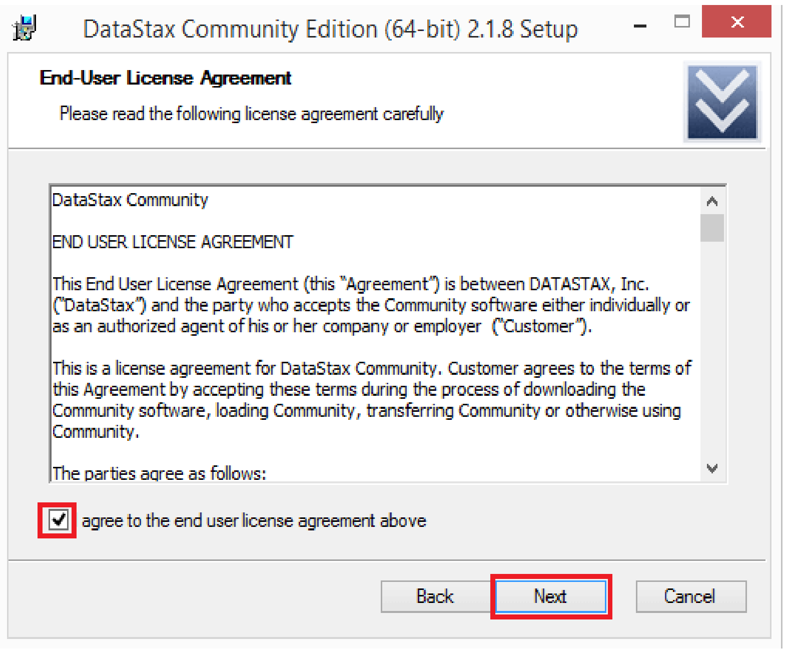

DataStax Community Edition (64-bit) 2.1.8 Setup

Welcome to the DataStax Community Edition (64-bit) 2.1.8 Setup

The setup will install DataStax Community Edition (64-bit) 2.1.8 on your computer. Click Next to continue or Cancel to exit.

This page gives you information about the Cassandra version you are going to install. Press the 'next' button.

Step 2)After pressing the 'next' button, following page will be displayed.


DataStax Community Edition (64-bit) 2.1.8 Setup

End-User License Agreement

Please read the following license agreement carefully

DataStax Community

END USER LICENSE AGREEMENT

This End User License Agreement (this "Agreement") is between DATASTAX, Inc. ("DataStax") and the party who accepts the Community software either individually or as an authorized agent of his or her company or employer ("Customer").

This is a license agreement for DataStax Community. Customer agrees to the terms of this Agreement by accepting these terms during the process of downloading the Community software, loading Community, transferring Community or otherwise using Community.

The parties agree as follows:

[✓] agree to the end user license agreement above

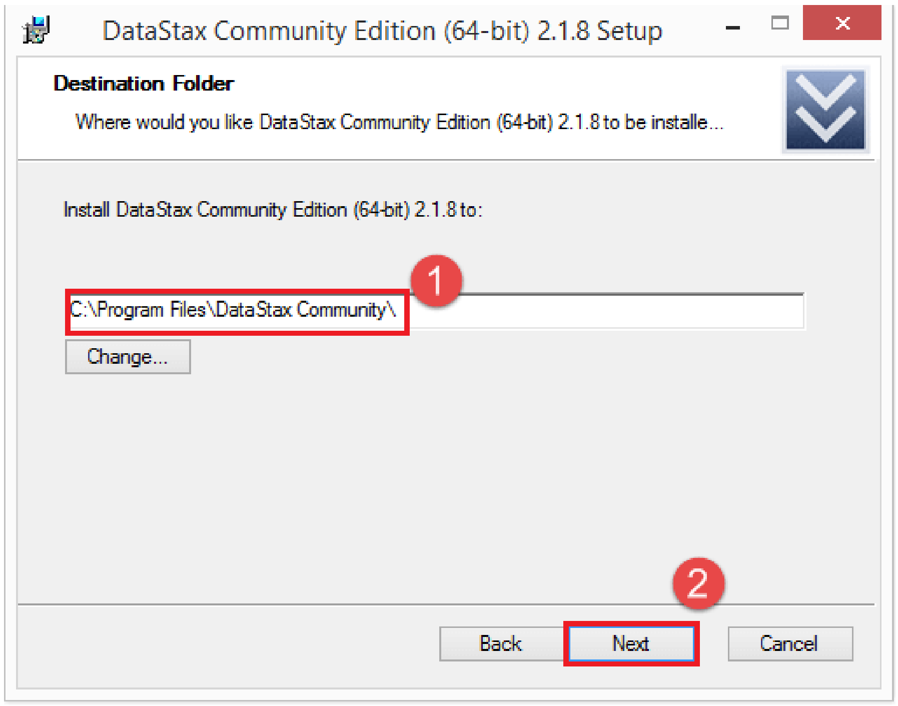
This page is about the license agreement. Mark the checkbox and press the next button.

Step 3) After pressing the 'next' button, the following page will be displayed.

This page asks about the installation location.

1. Default location is C:\Program Files. You can change installation location if you want to change. It is recommended not to change installation location.

2. After setting installation location, press the 'next' button



DataStax Community Edition (64-bit) 2.1.8 Setup

Destination Folder

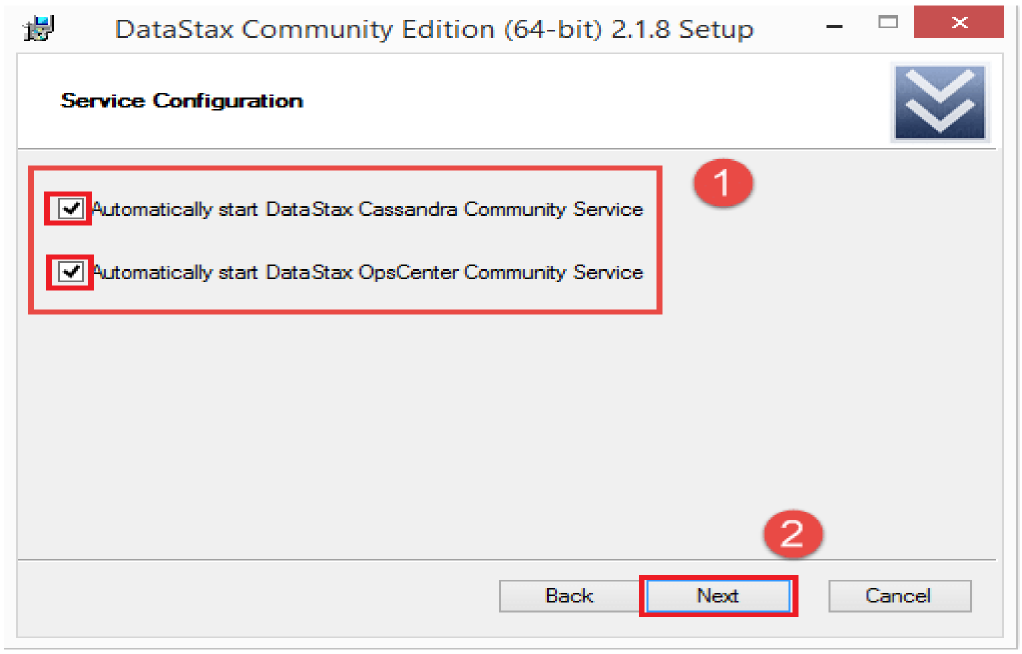
Where would you like DataStax Community Edition (64-bit) 2.1.8 to be installe...

Install DataStax Community Edition (64-bit) 2.1.8 to:

C:\Program Files\DataStax Community\

Step 4) After pressing 'next' button in above step, the following page will be displayed. This page asks about whether you want to automatically start Cassandra and OpsCenter.

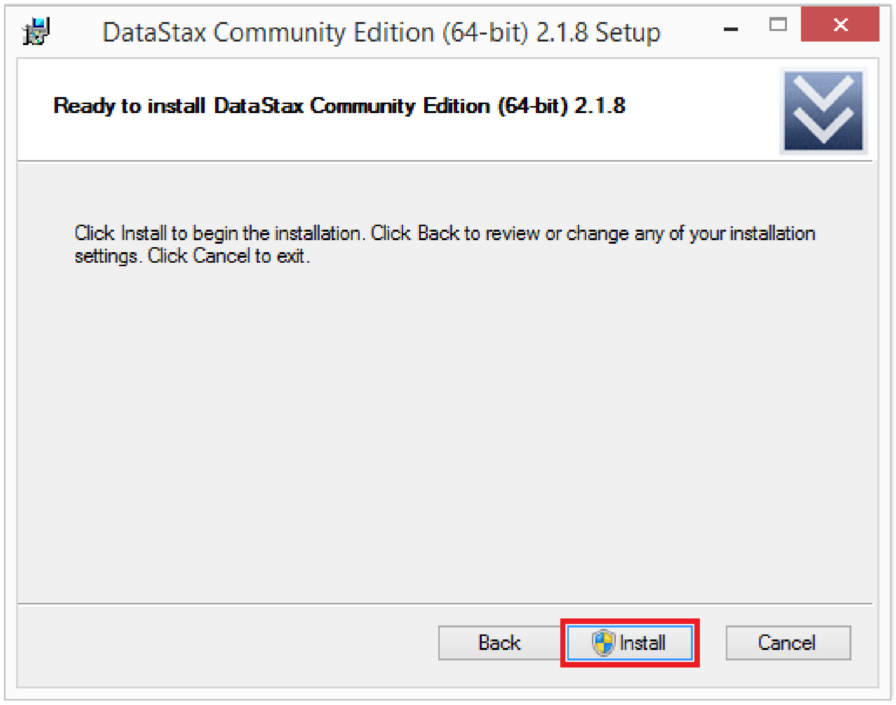
1. Mark the checkboxes if you to want to automatically start Cassandra and opsCenter.
2. After providing this information, press the 'next' button.


DataStax Community Edition (64-bit) 2.1.8 Setup

Service Configuration

- [✓] Automatically start DataStax Cassandra Community Service
- [✓] Automatically start DataStax OpsCenter Community Service

Step 5) After pressing the next button, following page will be displayed.

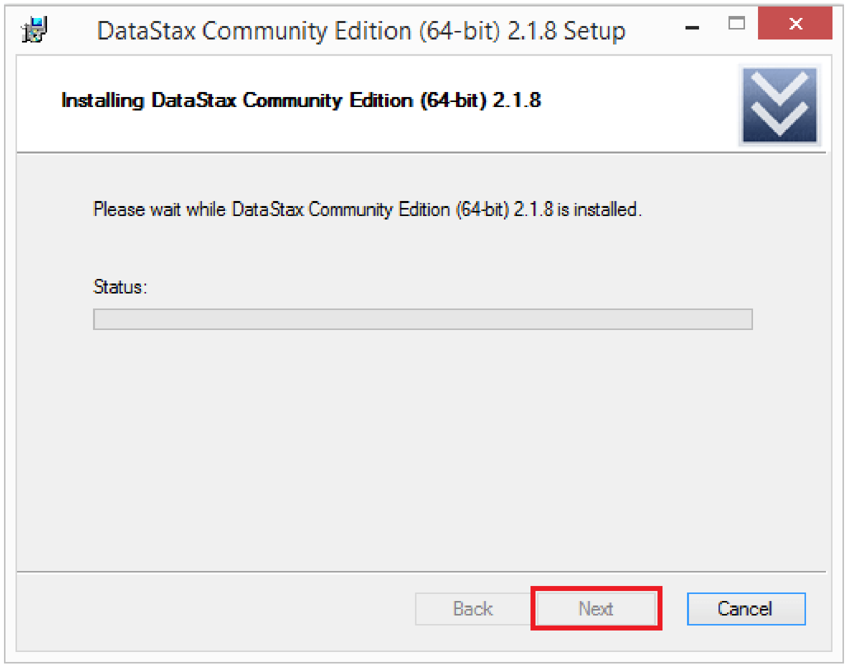

DataStax Community Edition (64-bit) 2.1.8 Setup

Ready to install DataStax Community Edition (64-bit) 2.1.8

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit.

Setup has collected all the necessary information and now the setup is ready to install. Press install button.

Step 6) After pressing 'install' button, following page will be displayed.

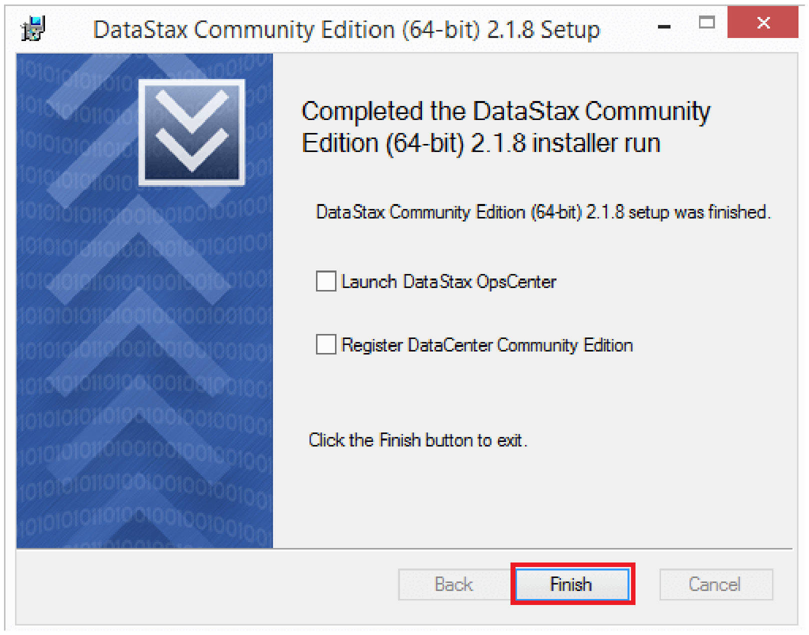

DataStax Community Edition (64-bit) 2.1.8 Setup

Installing DataStax Community Edition (64-bit) 2.1.8

Please wait while DataStax Community Edition (64-bit) 2.1.8 is installed.

Status:

Datastax community edition is being installed. After installation is completed, click on next button. When setup is installed successfully, press the 'Finish' button.


DataStax Community Edition (64-bit) 2.1.8 Setup

Completed the DataStax Community Edition (64-bit) 2.1.8 installer run

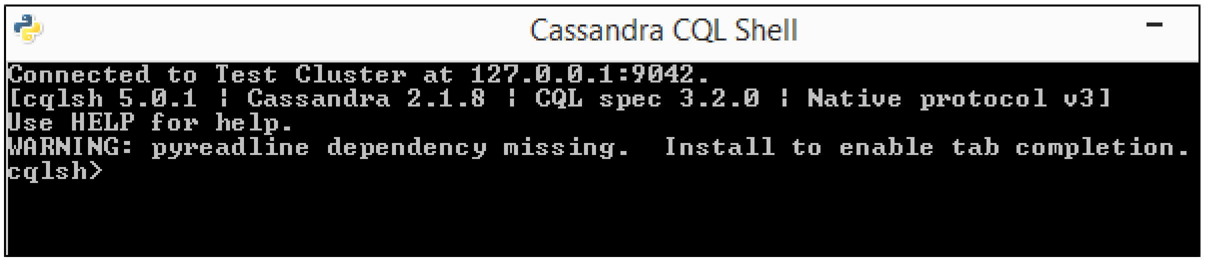
DataStax Community Edition (64-bit) 2.1.8 setup was finished.

☐ Launch DataStax OpsCenter

☐ Register DataCenter Community Edition

Click the Finish button to exit.

Completion phase of Cassandra Installation



```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.8 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh>
```

Now you can create a keyspace, tables, and write queries.

Steps ti Install MSI Link:

Download MSI here

https://docs.nomagic.com/display/CEDW183/Installing+and+configuring+Cassandra+on+Windows

Integration of PHP and Cassandra

Step 1: Download PHP Cassandra driver here

http://downloads.datastax.com/php-driver/windows/cassandra/v1.3.0/cassandra-php-driver-1.3.0-5.6-ts-vc11-x86.zip

Step 2: Copy and Paste PHP Cassandra driver to this folder location: C:\xampp\php\ext

Step 3: Open php.ini file and add the code below

[Cassandra]

extension=php_cassandra.dll

Step 4: Restart the Apache web Server

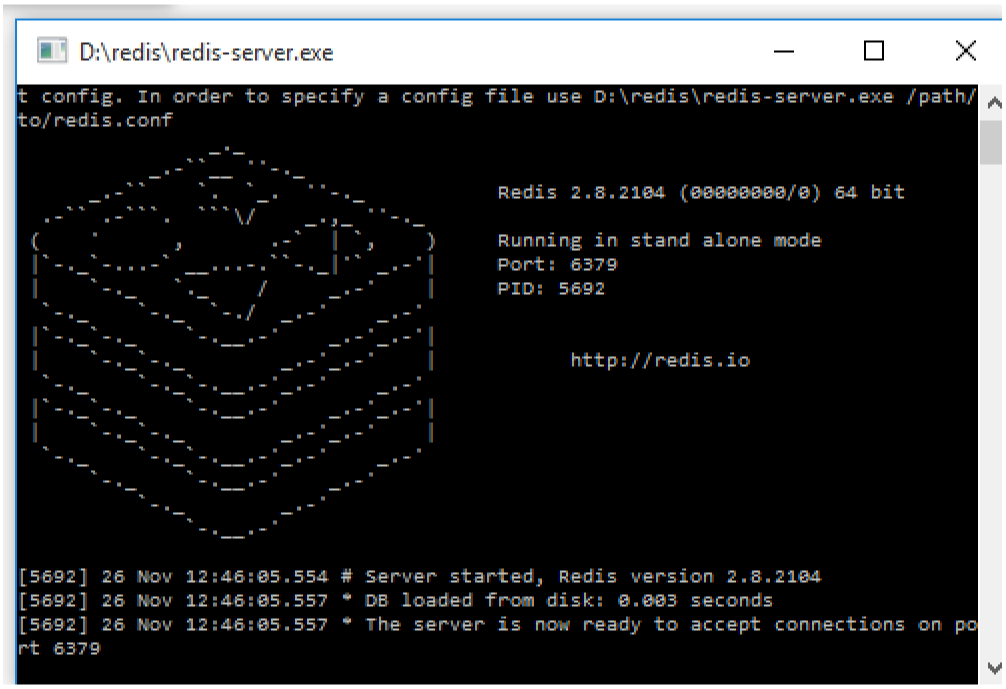
Steps to Install Redis

Step 1: Download Redis-x64-2.8.2104.zip

https://github.com/MSOpenTech/redis/releases/tag/win-2.8.2104

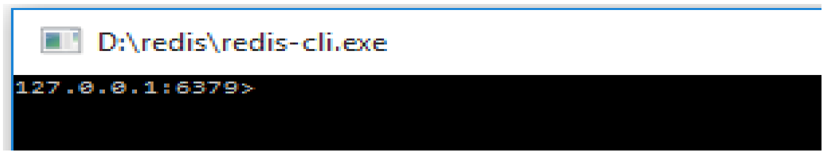
Step 2: Extract the zip to prepared directory

Step 3: Run redis-server.exe

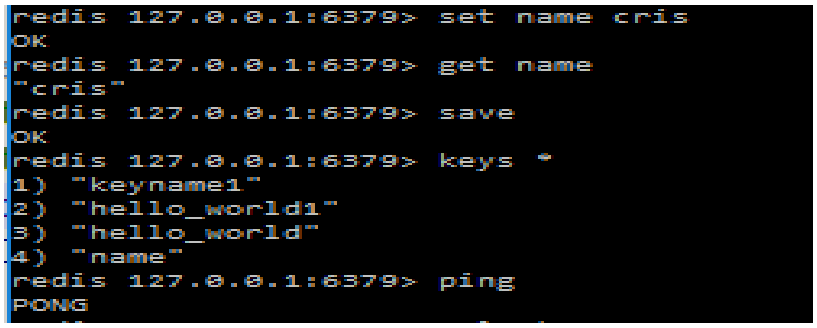


```
D:\redis\redis-server.exe
t config. In order to specify a config file use D:\redis\redis-server.exe /path/
to/redis.conf
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 2.8.2104 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in stand alone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 5692
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

[5692] 26 Nov 12:46:05.554 # Server started, Redis version 2.8.2104
[5692] 26 Nov 12:46:05.557 * DB loaded from disk: 0.003 seconds
[5692] 26 Nov 12:46:05.557 * The server is now ready to accept connections on po
rt 6379
```

Step 4: Run redis-cli.exe

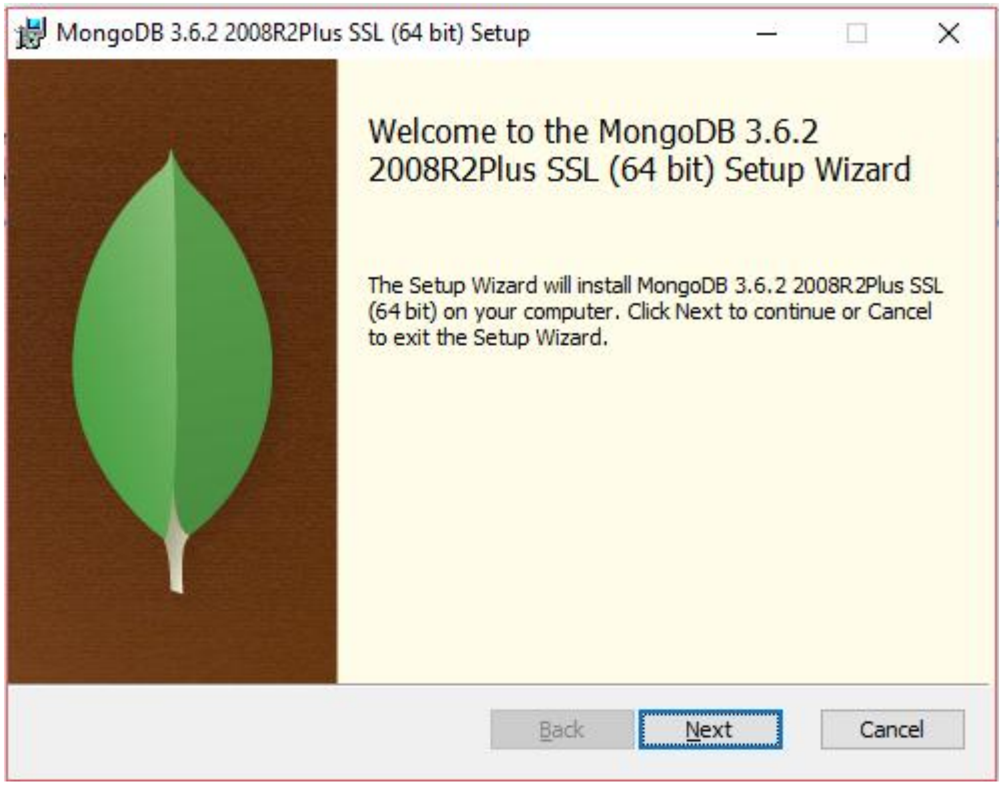


```
D:\redis\redis-cli.exe
127.0.0.1:6379>
```



```
redis 127.0.0.1:6379> set name cris
OK
redis 127.0.0.1:6379> get name
"cris"
redis 127.0.0.1:6379> save
OK
redis 127.0.0.1:6379> keys *
1) "keyname1"
2) "hello_world1"
3) "hello_world"
4) "name"
redis 127.0.0.1:6379> ping
PONG
```

Powered Redis Server

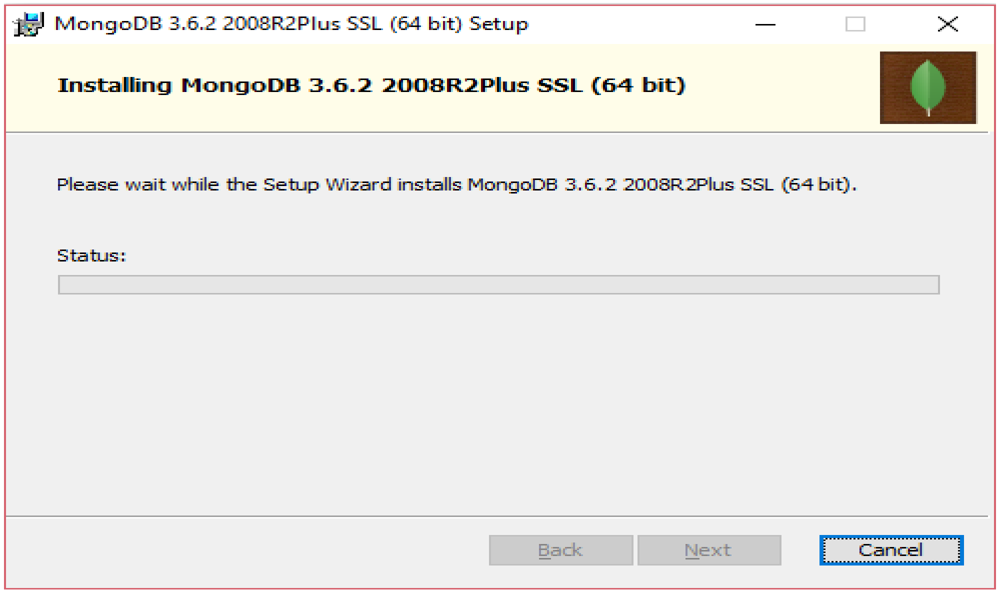


MongoDB 3.6.2 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 3.6.2 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 3.6.2 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

MongoDB installation Setup



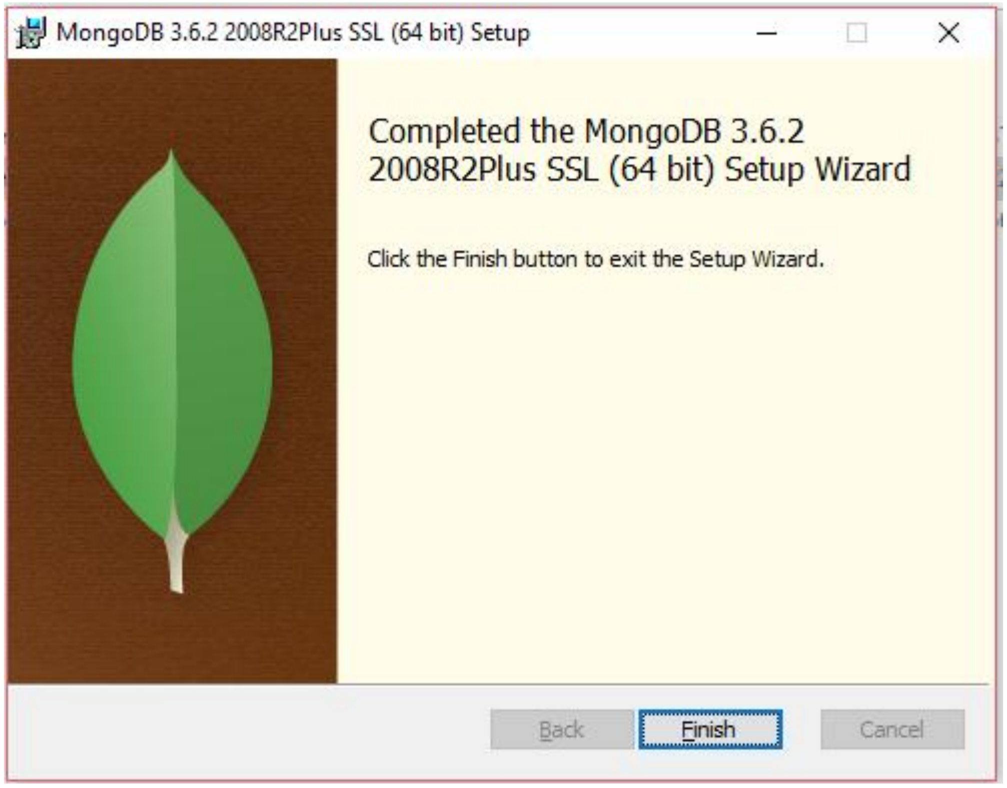
MongoDB 3.6.2 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 3.6.2 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 3.6.2 2008R2Plus SSL (64 bit).

Status:

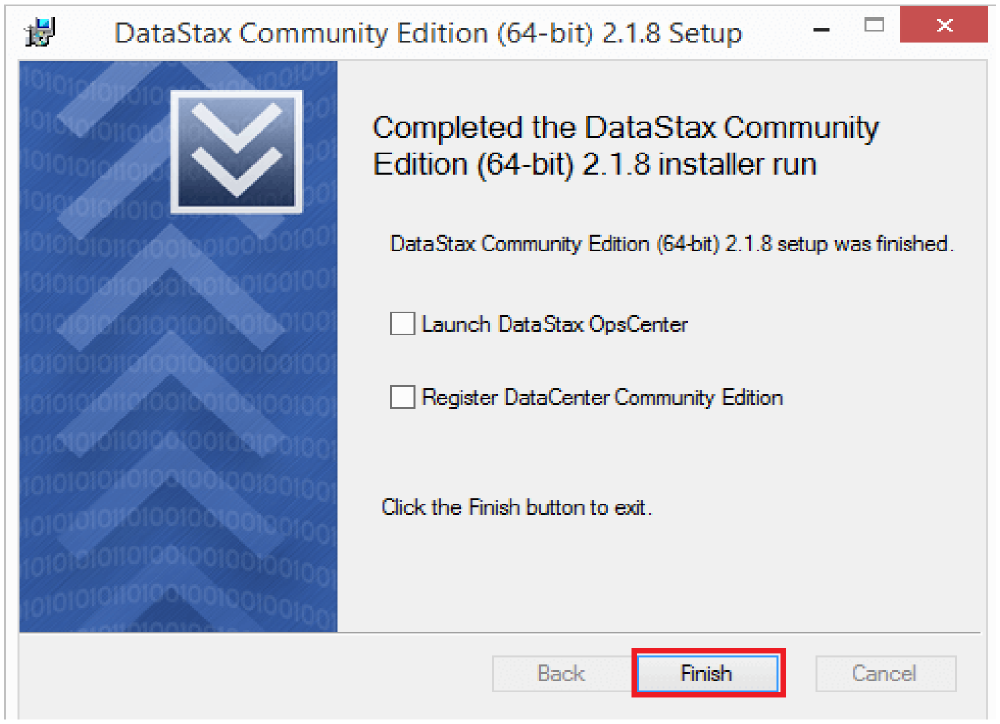
Mongodb Installation in progress


MongoDB 3.6.2 2008R2Plus SSL (64 bit) Setup

Completed the MongoDB 3.6.2 2008R2Plus SSL (64 bit) Setup Wizard

Click the Finish button to exit the Setup Wizard.

Complete installation of the MongoDB software


DataStax Community Edition (64-bit) 2.1.8 Setup

Completed the DataStax Community Edition (64-bit) 2.1.8 installer run

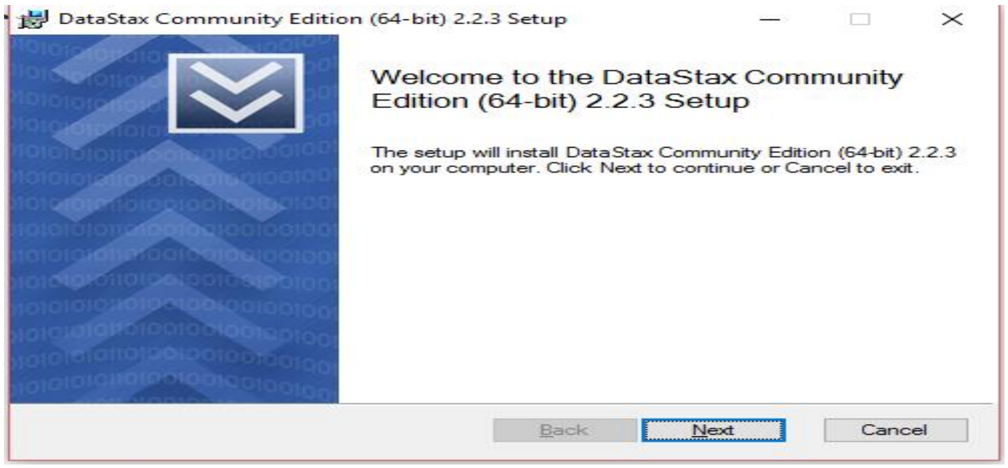
DataStax Community Edition (64-bit) 2.1.8 setup was finished.

☐ Launch DataStax OpsCenter

☐ Register DataCenter Community Edition

Click the Finish button to exit.

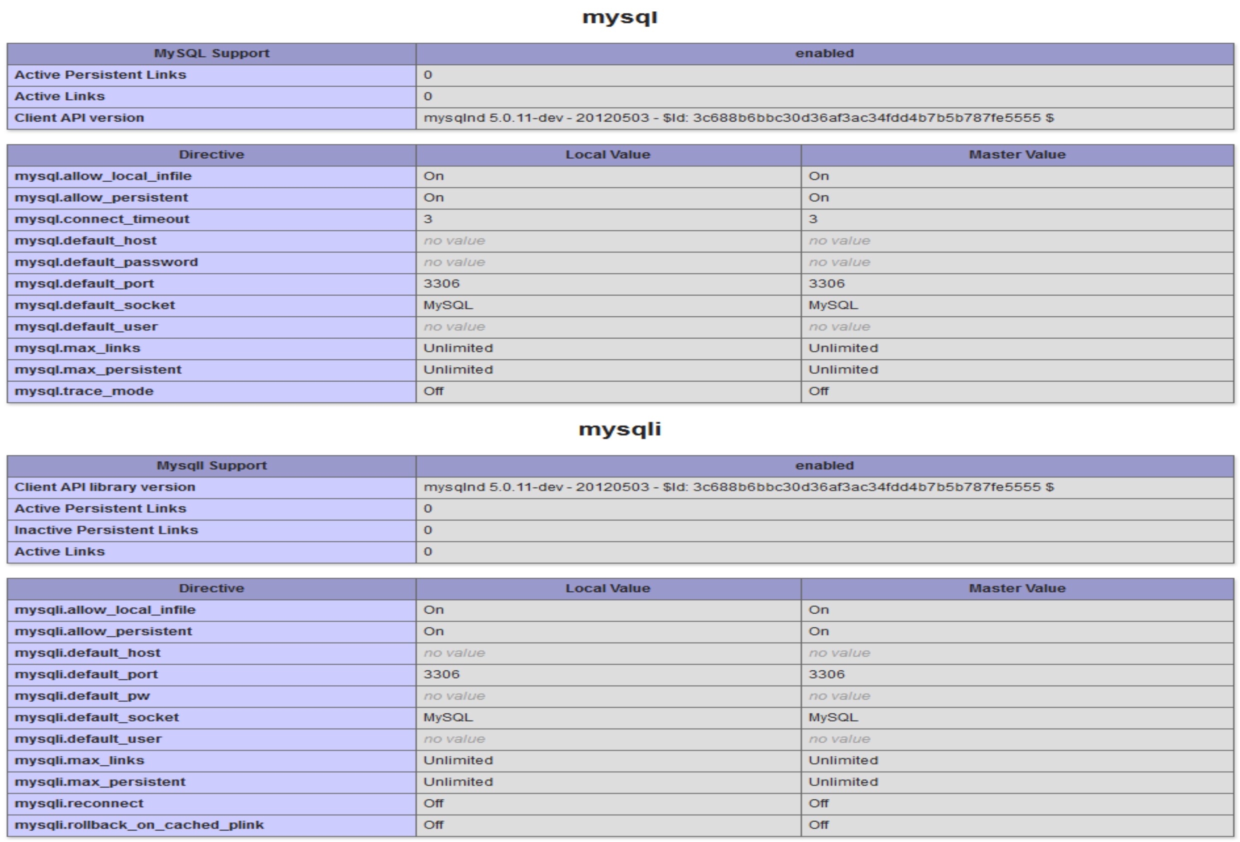
Complete installation of the DataStax (Cassandra) software



Welcome to the DataStax Community Edition (64-bit) 2.2.3 Setup

The setup will install DataStax Community Edition (64-bit) 2.2.3 on your computer. Click Next to continue or Cancel to exit.

Welcome screen for the software installation of Cassandra NoSQL.



mysql

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	mysqlnd 5.0.11-dev - 20120503 - $Id: 3c688b6bbc30d36af3ac34fdd4b7b5b787fe5555 $

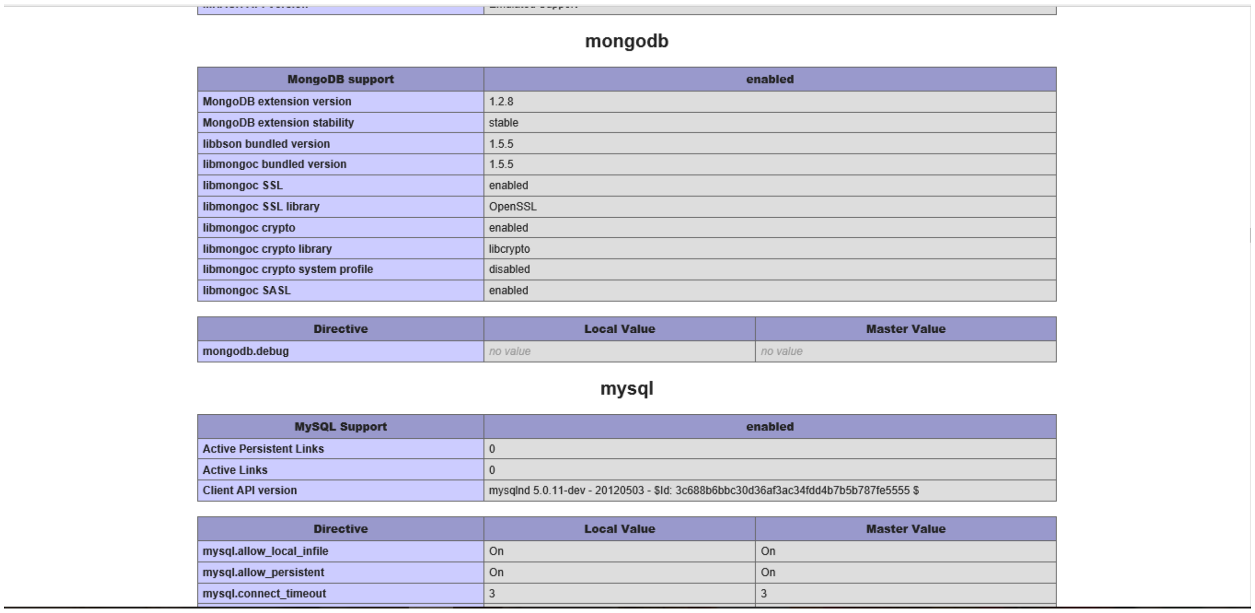
Directive	Local Value	Master Value
mysql.allow_local_infile	On	On
mysql.allow_persistent	On	On
mysql.connect_timeout	3	3
mysql.default_host	*no value*	*no value*
mysql.default_password	*no value*	*no value*
mysql.default_port	3306	3306
mysql.default_socket	MySQL	MySQL
mysql.default_user	*no value*	*no value*
mysql.max_links	Unlimited	Unlimited
mysql.max_persistent	Unlimited	Unlimited
mysql.trace_mode	Off	Off

mysqli

MysqlI Support	enabled
Client API library version	mysqlnd 5.0.11-dev - 20120503 - $Id: 3c688b6bbc30d36af3ac34fdd4b7b5b787fe5555 $
Active Persistent Links	0
Inactive Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
mysqli.allow_local_infile	On	On
mysqli.allow_persistent	On	On
mysqli.default_host	*no value*	*no value*
mysqli.default_port	3306	3306
mysqli.default_pw	*no value*	*no value*
mysqli.default_socket	MySQL	MySQL
mysqli.default_user	*no value*	*no value*
mysqli.max_links	Unlimited	Unlimited
mysqli.max_persistent	Unlimited	Unlimited
mysqli.reconnect	Off	Off
mysqli.rollback_on_cached_plink	Off	Off

Php mysql integration



mongodb

MongoDB support	enabled
MongoDB extension version	1.2.8
MongoDB extension stability	stable
libbson bundled version	1.5.5
libmongoc bundled version	1.5.5
libmongoc SSL	enabled
libmongoc SSL library	OpenSSL
libmongoc crypto	enabled
libmongoc crypto library	libcrypto
libmongoc crypto system profile	disabled
libmongoc SASL	enabled

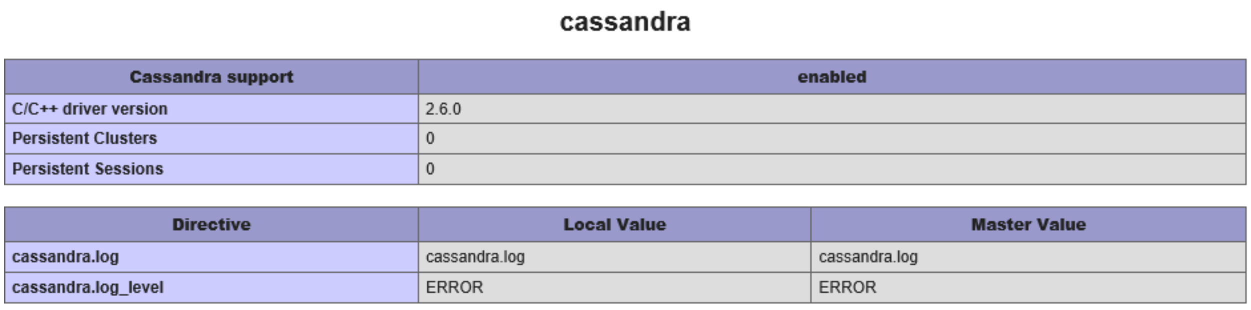
Directive	Local Value	Master Value
mongodb.debug	*no value*	*no value*

mysql

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	mysqlnd 5.0.11-dev - 20120503 - $Id: 3c688b6bbc30d36af3ac34fdd4b7b5b787fe5555 $

Directive	Local Value	Master Value
mysql.allow_local_infile	On	On
mysql.allow_persistent	On	On
mysql.connect_timeout	3	3

PHP MongoDB Integration



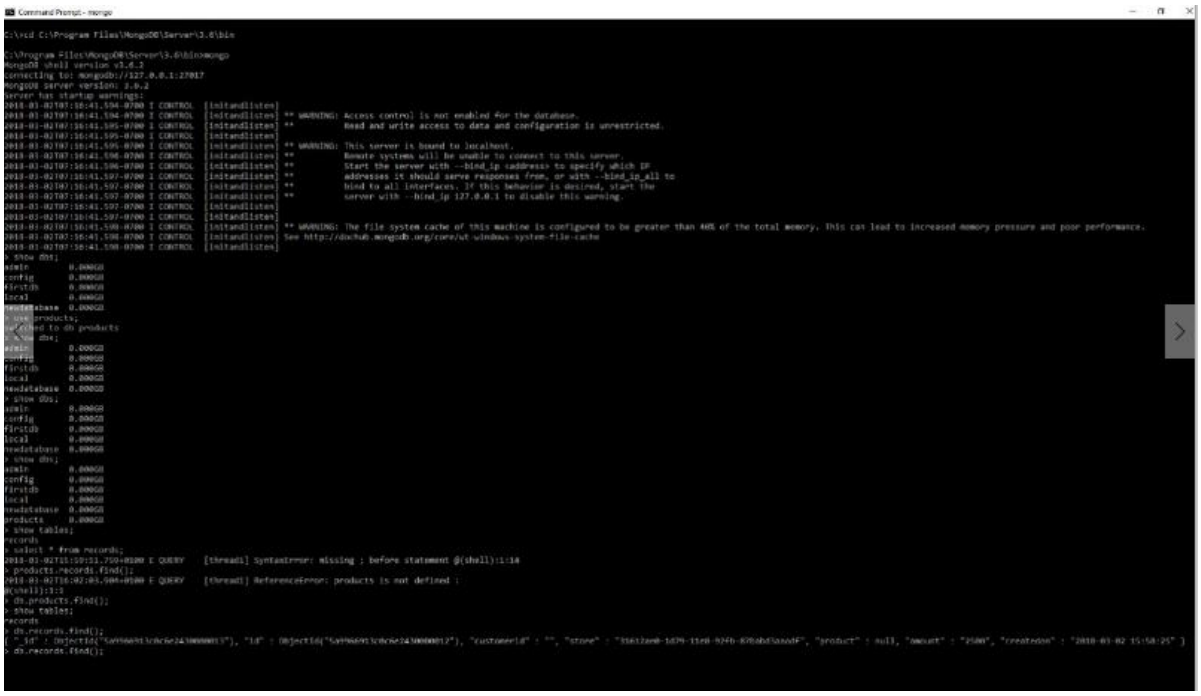
cassandra

Cassandra support	enabled
C/C++ driver version	2.6.0
Persistent Clusters	0
Persistent Sessions	0

Directive	Local Value	Master Value
cassandra.log	cassandra.log	cassandra.log
cassandra.log_level	ERROR	ERROR

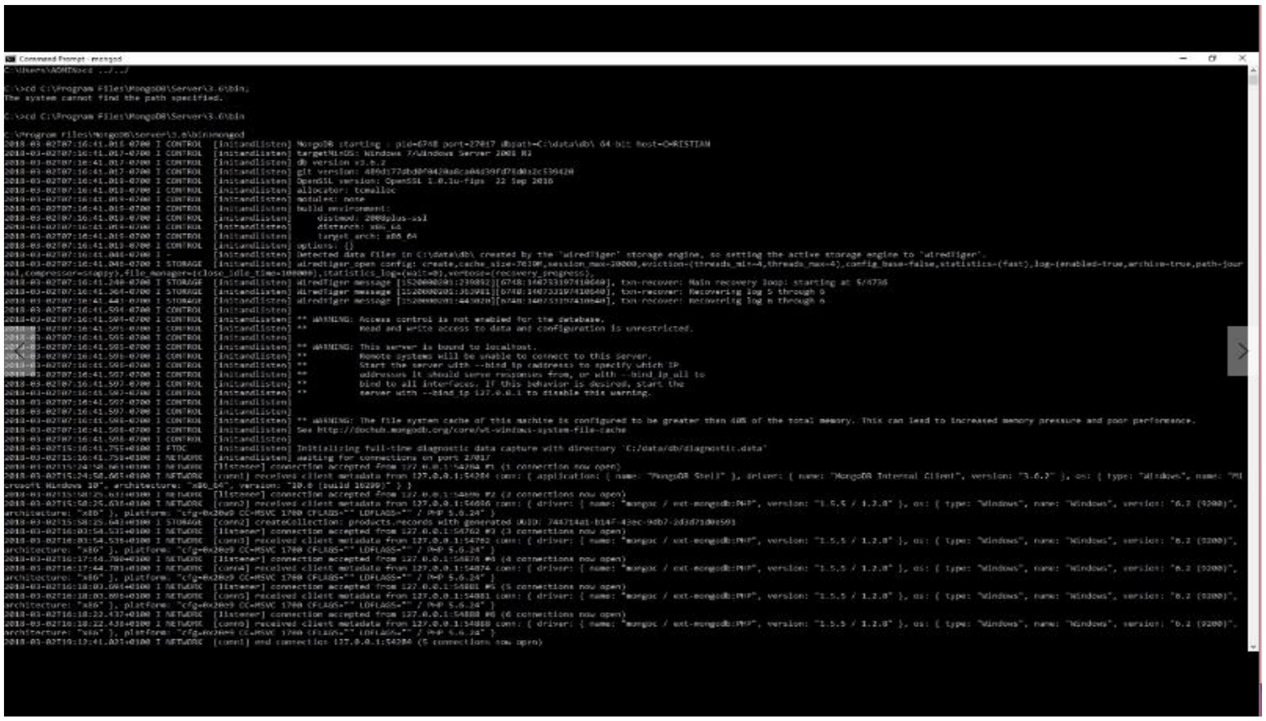
PHP Cassandra Integration

To initiate saving information to several other database engines.



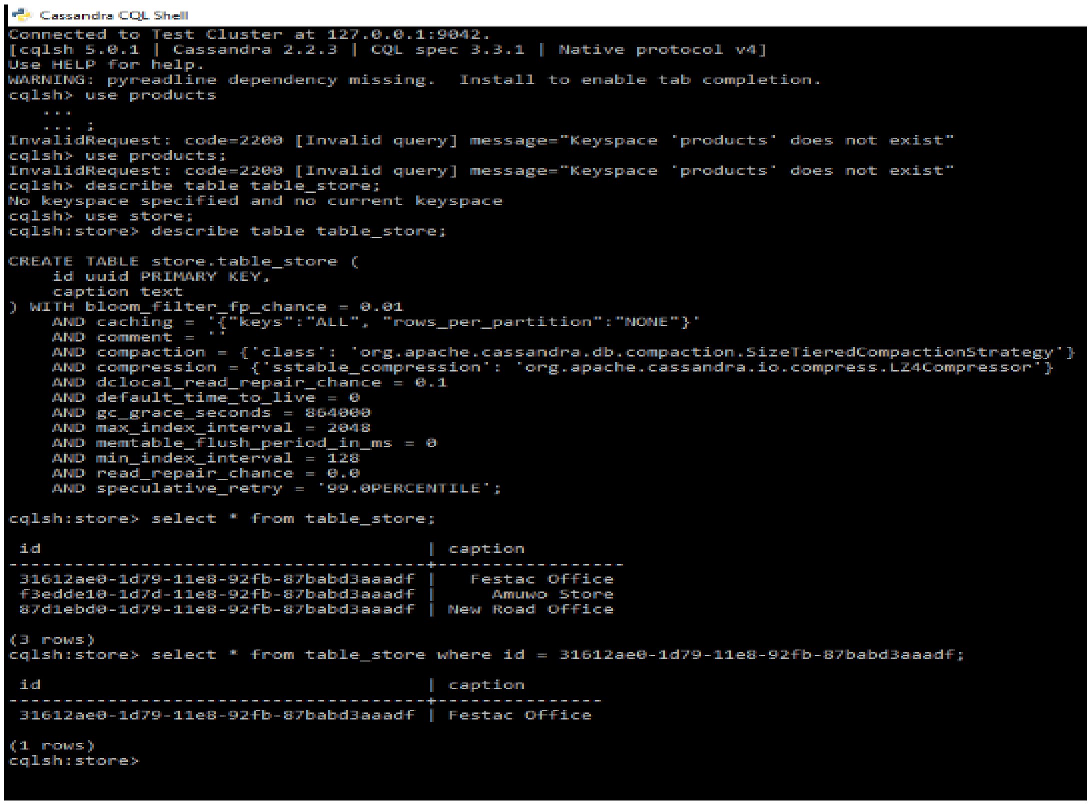
```
Command Prompt - mongo

C:\>cd C:\Program Files\MongoDB\Server\3.6\bin

C:\Program Files\MongoDB\Server\3.6\bin>mongo
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
Server has startup warnings:
2018-03-02T07:16:41.594-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.594-0700 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-03-02T07:16:41.596-0700 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this server.
2018-03-02T07:16:41.596-0700 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify which IP
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --bind_ip_all to
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired, start the
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warning.
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten] ** WARNING: The file system cache of this machine is configured to be greater than 40% of the total memory. This can lead to increased memory pressure and poor performance.
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten] See http://dochub.mongodb.org/core/wt-windows-system-file-cache
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten]
> show dbs;
admin        0.000GB
config       0.000GB
firstdb      0.000GB
local        0.000GB
newdatabase  0.000GB
> use products;
switched to db products
> show dbs;
admin        0.000GB
config       0.000GB
firstdb      0.000GB
local        0.000GB
newdatabase  0.000GB
> show dbs;
admin        0.000GB
config       0.000GB
firstdb      0.000GB
local        0.000GB
newdatabase  0.000GB
> show dbs;
admin        0.000GB
config       0.000GB
firstdb      0.000GB
local        0.000GB
newdatabase  0.000GB
products     0.000GB
> show tables;
records
> select * from records;
2018-03-02T15:59:51.759+0100 E QUERY    [thread1] SyntaxError: missing ; before statement @(shell):1:14
> products.records.find();
2018-03-02T16:02:03.984+0100 E QUERY    [thread1] ReferenceError: products is not defined :
@(shell):1:1
> db.products.find();
> show tables;
records
> db.records.find();
{ "_id" : ObjectId("5a99669313c0c6e243000013"), "id" : ObjectId("5a99669313c0c6e243000012"), "customerid" : "", "store" : "31612ae0-1d79-11e8-924b-878abd3aaadf", "product" : null, "amount" : "2500", "createdon" : "2018-03-02 15:58:25" }
> db.records.find();
```

Command Line Interface Displaying MongoDB Server Starting Up



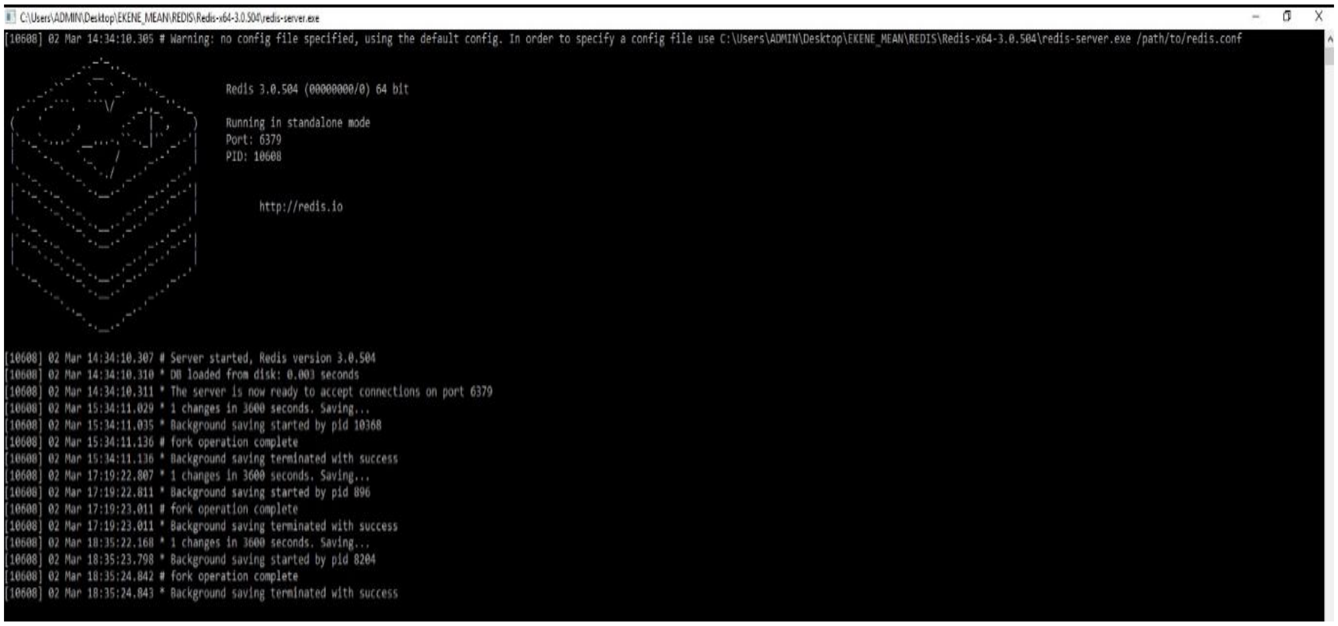
```
Command Prompt - mongod
C:\Users\ADMIN>cd ../../

C:\>cd C:\Program Files\MongoDB\Server\3.6\bin;
The system cannot find the path specified.

C:\>cd C:\Program Files\MongoDB\Server\3.6\bin

C:\Program Files\MongoDB\Server\3.6\bin>mongod
2018-03-02T07:16:41.016-0700 I CONTROL  [initandlisten] MongoDB starting : pid=6748 port=27017 dbpath=C:\data\db\ 64-bit host=CHRISTIAN
2018-03-02T07:16:41.017-0700 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-03-02T07:16:41.017-0700 I CONTROL  [initandlisten] db version v3.6.2
2018-03-02T07:16:41.017-0700 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a0c aa6e39fd78d8a2c539420
2018-03-02T07:16:41.018-0700 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips  22 Sep 2016
2018-03-02T07:16:41.018-0700 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten] modules: none
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten] build environment:
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten]     distmod: 2008plus-ssl
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-02T07:16:41.019-0700 I CONTROL  [initandlisten] options: {}
2018-03-02T07:16:41.046-0700 I -        [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2018-03-02T07:16:41.046-0700 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=7618M,session_max=20000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=jour
nal,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress),
2018-03-02T07:16:41.240-0700 I STORAGE  [initandlisten] WiredTiger message [1520000201:239892][6748:140733197410640], txn-recover: Main recovery loop: starting at 5/4736
2018-03-02T07:16:41.364-0700 I STORAGE  [initandlisten] WiredTiger message [1520000201:363981][6748:140733197410640], txn-recover: Recovering log 5 through 6
2018-03-02T07:16:41.443-0700 I STORAGE  [initandlisten] WiredTiger message [1520000201:443020][6748:140733197410640], txn-recover: Recovering log 6 through 6
2018-03-02T07:16:41.594-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.594-0700 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.595-0700 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-03-02T07:16:41.596-0700 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this server.
2018-03-02T07:16:41.596-0700 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify which IP
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --bind_ip_all to
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired, start the
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warning.
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.597-0700 I CONTROL  [initandlisten]
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten] ** WARNING: The file system cache of this machine is configured to be greater than 40% of the total memory. This can lead to increased memory pressure and poor performance.
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten] See http://dochub.mongodb.org/core/wt-windows-system-file-cache
2018-03-02T07:16:41.598-0700 I CONTROL  [initandlisten]
2018-03-02T15:16:41.755+0100 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/data/db/diagnostic.data'
2018-03-02T15:16:41.758+0100 I NETWORK  [initandlisten] waiting for connections on port 27017
2018-03-02T15:24:58.661+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54284 #1 (1 connection now open)
2018-03-02T15:24:58.665+0100 I NETWORK  [conn1] received client metadata from 127.0.0.1:54284 conn: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.6.2" }, os: { type: "Windows", name: "Mi
crosoft Windows 10", architecture: "x86_64", version: "10.0 (build 16299)" } }
2018-03-02T15:58:25.633+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54686 #2 (2 connections now open)
2018-03-02T15:58:25.638+0100 I NETWORK  [conn2] received client metadata from 127.0.0.1:54686 conn: { driver: { name: "mongoc / ext-mongodb:PHP", version: "1.5.5 / 1.2.8" }, os: { type: "Windows", name: "Windows", version: "6.2 (9200)",
architecture: "x86" }, platform: "cfg=0x20e9 CC=MSVC 1700 CFLAGS="" LDFLAGS="" / PHP 5.6.24" }
2018-03-02T15:58:25.643+0100 I STORAGE  [conn2] createCollection: products.records with generated UUID: 744714a1-b14f-43ec-94b7-2d3d71d0e591
2018-03-02T16:03:54.535+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54762 #3 (3 connections now open)
2018-03-02T16:03:54.538+0100 I NETWORK  [conn3] received client metadata from 127.0.0.1:54762 conn: { driver: { name: "mongoc / ext-mongodb:PHP", version: "1.5.5 / 1.2.8" }, os: { type: "Windows", name: "Windows", version: "6.2 (9200)",
architecture: "x86" }, platform: "cfg=0x20e9 CC=MSVC 1700 CFLAGS="" LDFLAGS="" / PHP 5.6.24" }
2018-03-02T16:17:44.780+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54874 #4 (4 connections now open)
2018-03-02T16:17:44.781+0100 I NETWORK  [conn4] received client metadata from 127.0.0.1:54874 conn: { driver: { name: "mongoc / ext-mongodb:PHP", version: "1.5.5 / 1.2.8" }, os: { type: "Windows", name: "Windows", version: "6.2 (9200)",
architecture: "x86" }, platform: "cfg=0x20e9 CC=MSVC 1700 CFLAGS="" LDFLAGS="" / PHP 5.6.24" }
2018-03-02T16:18:03.694+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54881 #5 (5 connections now open)
2018-03-02T16:18:03.695+0100 I NETWORK  [conn5] received client metadata from 127.0.0.1:54881 conn: { driver: { name: "mongoc / ext-mongodb:PHP", version: "1.5.5 / 1.2.8" }, os: { type: "Windows", name: "Windows", version: "6.2 (9200)",
architecture: "x86" }, platform: "cfg=0x20e9 CC=MSVC 1700 CFLAGS="" LDFLAGS="" / PHP 5.6.24" }
2018-03-02T16:18:22.437+0100 I NETWORK  [listener] connection accepted from 127.0.0.1:54888 #6 (6 connections now open)
2018-03-02T16:18:22.438+0100 I NETWORK  [conn6] received client metadata from 127.0.0.1:54888 conn: { driver: { name: "mongoc / ext-mongodb:PHP", version: "1.5.5 / 1.2.8" }, os: { type: "Windows", name: "Windows", version: "6.2 (9200)",
architecture: "x86" }, platform: "cfg=0x20e9 CC=MSVC 1700 CFLAGS="" LDFLAGS="" / PHP 5.6.24" }
2018-03-02T19:12:41.027+0100 I NETWORK  [conn1] end connection 127.0.0.1:54284 (5 connections now open)
```

Command Line Interface Displaying MongoDB Shell



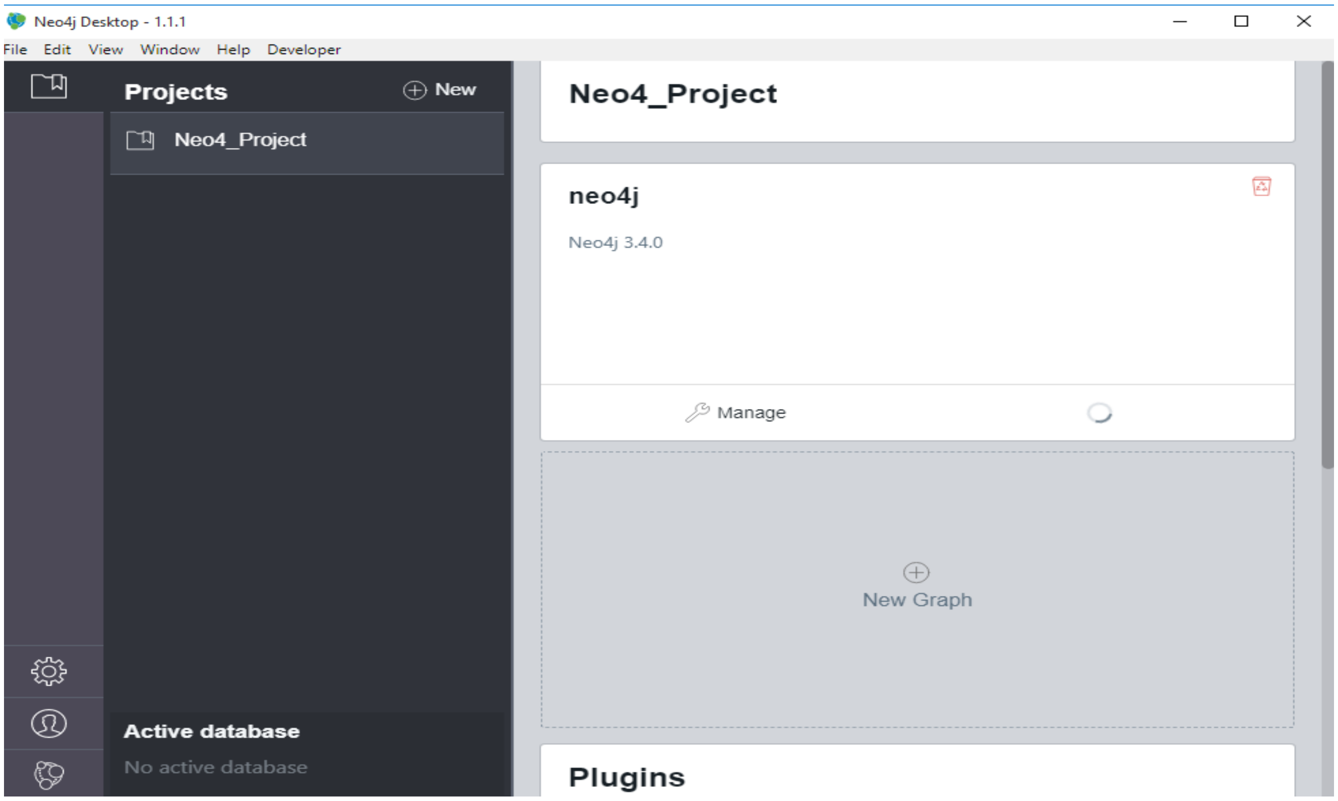
```
Cassandra CQL Shell
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> use products
   ...
   ... ;
InvalidRequest: code=2200 [Invalid query] message="Keyspace 'products' does not exist"
cqlsh> use products;
InvalidRequest: code=2200 [Invalid query] message="Keyspace 'products' does not exist"
cqlsh> describe table table_store;
No keyspace specified and no current keyspace
cqlsh> use store;
cqlsh:store> describe table table_store;

CREATE TABLE store.table_store (
    id uuid PRIMARY KEY,
    caption text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';

cqlsh:store> select * from table_store;

 id                                   | caption
--------------------------------------+-----------------
 31612ae0-1d79-11e8-92fb-87babd3aaadf |   Festac Office
 f3edde10-1d7d-11e8-92fb-87babd3aaadf |     Amuwo Store
 87d1ebd0-1d79-11e8-92fb-87babd3aaadf | New Road Office

(3 rows)
cqlsh:store> select * from table_store where id = 31612ae0-1d79-11e8-92fb-87babd3aaadf;

 id                                   | caption
--------------------------------------+---------------
 31612ae0-1d79-11e8-92fb-87babd3aaadf | Festac Office

(1 rows)
cqlsh:store>
```

Command Line Interface displaying Cassandra Shell and Sample Queries

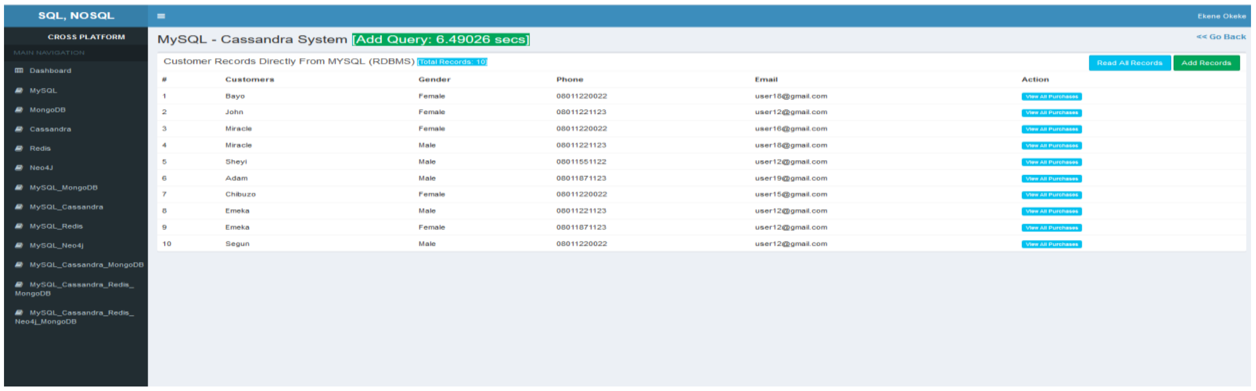


```
C:\Users\ADMIN\Desktop\EKENE_MEAN\REDIS\Redis-x64-3.0.504\redis-server.exe
[10608] 02 Mar 14:34:10.305 # Warning: no config file specified, using the default config. In order to specify a config file use C:\Users\ADMIN\Desktop\EKENE_MEAN\REDIS\Redis-x64-3.0.504\redis-server.exe /path/to/redis.conf

                Redis 3.0.504 (00000000/0) 64 bit

                Running in standalone mode
                Port: 6379
                PID: 10608

                     http://redis.io

[10608] 02 Mar 14:34:10.307 # Server started, Redis version 3.0.504
[10608] 02 Mar 14:34:10.310 * DB loaded from disk: 0.003 seconds
[10608] 02 Mar 14:34:10.311 * The server is now ready to accept connections on port 6379
[10608] 02 Mar 15:34:11.029 * 1 changes in 3600 seconds. Saving...
[10608] 02 Mar 15:34:11.035 * Background saving started by pid 10368
[10608] 02 Mar 15:34:11.136 # fork operation complete
[10608] 02 Mar 15:34:11.136 * Background saving terminated with success
[10608] 02 Mar 17:19:22.807 * 1 changes in 3600 seconds. Saving...
[10608] 02 Mar 17:19:22.811 * Background saving started by pid 896
[10608] 02 Mar 17:19:23.011 # fork operation complete
[10608] 02 Mar 17:19:23.011 * Background saving terminated with success
[10608] 02 Mar 18:35:22.168 * 1 changes in 3600 seconds. Saving...
[10608] 02 Mar 18:35:23.798 * Background saving started by pid 8204
[10608] 02 Mar 18:35:24.842 # fork operation complete
[10608] 02 Mar 18:35:24.843 * Background saving terminated with success
```

Command Line Interface displaying Redis Server starting up



Powered Neo4j Server

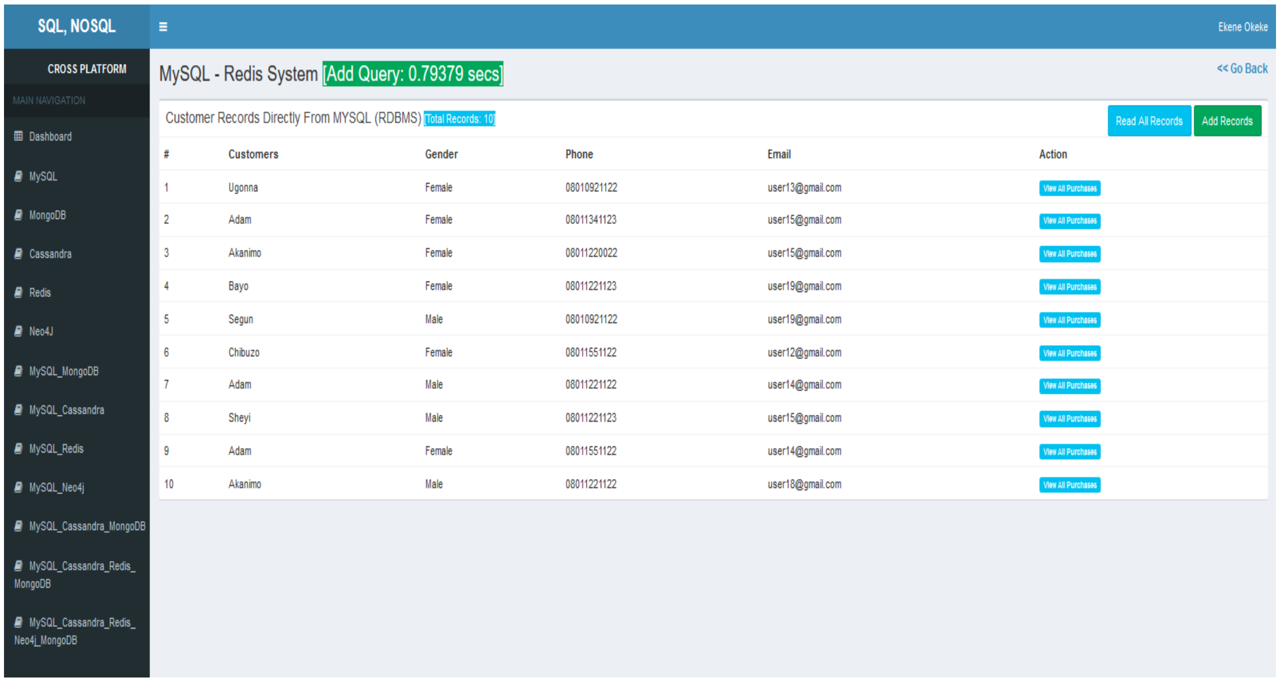


MySQL - Cassandra System [Add Query: 6.49026 secs]

Customer Records Directly From MYSQL (RDBMS) [Total Records: 10]

#	Customers	Gender	Phone	Email	Action
1	Bayo	Female	08011220022	user18@gmail.com	View All Purchases
2	John	Female	08011221123	user12@gmail.com	View All Purchases
3	Miracle	Female	08011220022	user16@gmail.com	View All Purchases
4	Miracle	Male	08011221123	user18@gmail.com	View All Purchases
5	Sheyi	Male	08011551122	user12@gmail.com	View All Purchases
6	Adam	Male	08011871123	user19@gmail.com	View All Purchases
7	Chibuzo	Female	08011220022	user15@gmail.com	View All Purchases
8	Emeka	Male	08011221123	user12@gmail.com	View All Purchases
9	Emeka	Female	08011871123	user12@gmail.com	View All Purchases
10	Segun	Male	08011220022	user12@gmail.com	View All Purchases

Input operation for MySQL_Cassandra

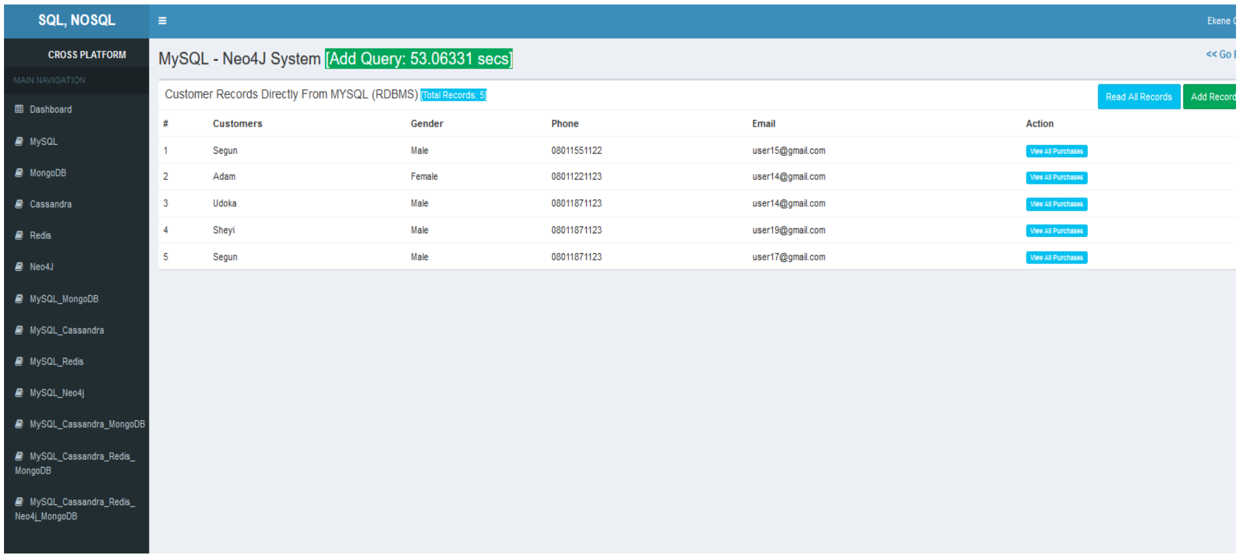


MySQL - Redis System [Add Query: 0.79379 secs]

Customer Records Directly From MYSQL (RDBMS) [Total Records: 10]

#	Customers	Gender	Phone	Email	Action
1	Ugonna	Female	08010921122	user13@gmail.com	View All Purchases
2	Adam	Female	08011341123	user15@gmail.com	View All Purchases
3	Akanimo	Female	08011220022	user15@gmail.com	View All Purchases
4	Bayo	Female	08011221123	user19@gmail.com	View All Purchases
5	Segun	Male	08010921122	user19@gmail.com	View All Purchases
6	Chibuzo	Female	08011551122	user12@gmail.com	View All Purchases
7	Adam	Male	08011221122	user14@gmail.com	View All Purchases
8	Sheyi	Male	08011221123	user15@gmail.com	View All Purchases
9	Adam	Female	08011551122	user14@gmail.com	View All Purchases
10	Akanimo	Male	08011221122	user18@gmail.com	View All Purchases

Input operation for MySQL_Redis

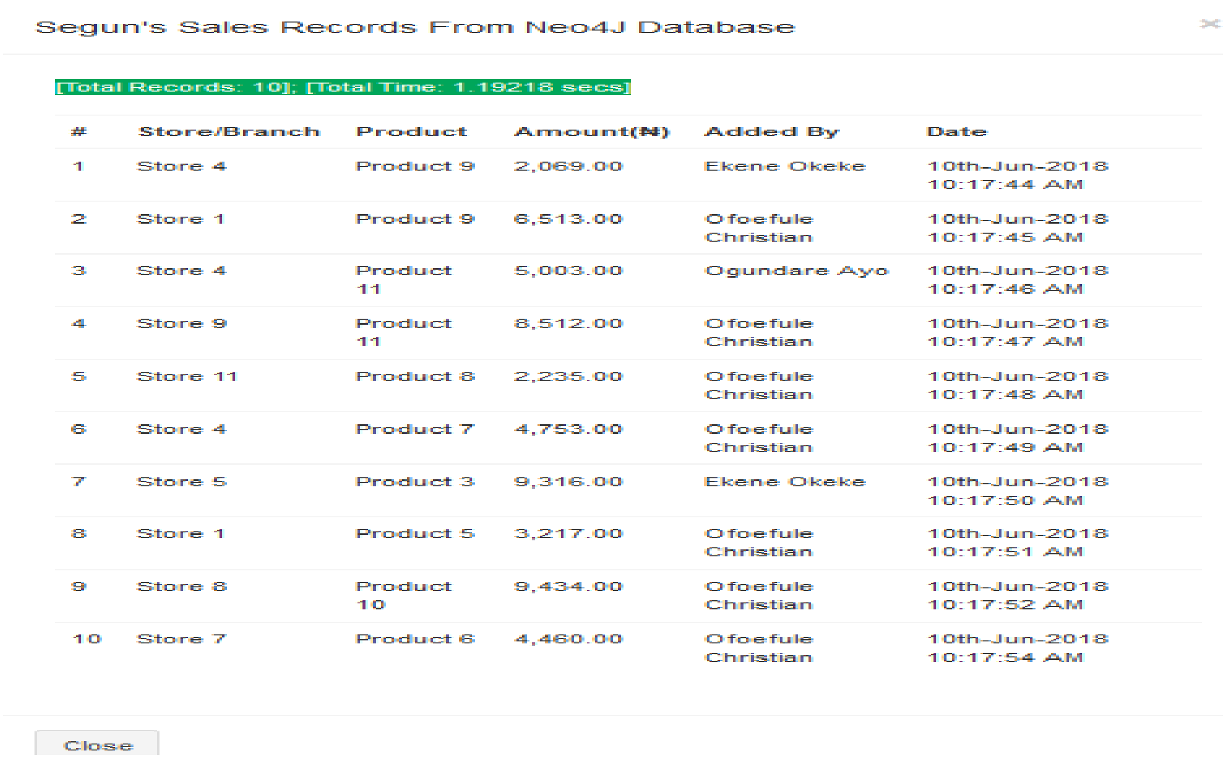


MySQL - Neo4J System [Add Query: 53.06331 secs]

Customer Records Directly From MYSQL (RDBMS) [Total Records: 5]

#	Customers	Gender	Phone	Email	Action
1	Segun	Male	08011551122	user15@gmail.com	View All Purchases
2	Adam	Female	08011221123	user14@gmail.com	View All Purchases
3	Udoka	Male	08011871123	user14@gmail.com	View All Purchases
4	Sheyi	Male	08011871123	user19@gmail.com	View All Purchases
5	Segun	Male	08011871123	user17@gmail.com	View All Purchases

Input operation for MySQL_Neo4j

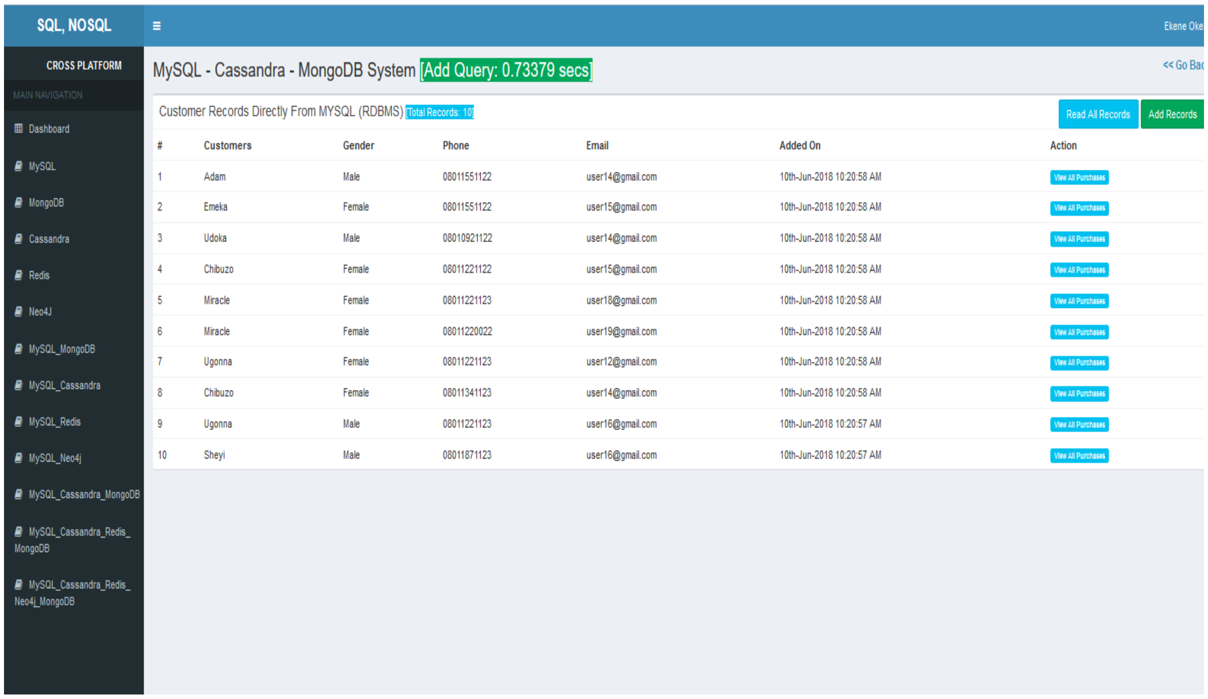


Segun's Sales Records From Neo4J Database

[Total Records: 10]; [Total Time: 1.19218 secs]

#	Store/Branch	Product	Amount(₦)	Added By	Date
1	Store 4	Product 9	2,069.00	Ekene Okeke	10th-Jun-2018 10:17:44 AM
2	Store 1	Product 9	6,513.00	Ofoefule Christian	10th-Jun-2018 10:17:45 AM
3	Store 4	Product 11	5,003.00	Ogundare Ayo	10th-Jun-2018 10:17:46 AM
4	Store 9	Product 11	8,512.00	Ofoefule Christian	10th-Jun-2018 10:17:47 AM
5	Store 11	Product 8	2,235.00	Ofoefule Christian	10th-Jun-2018 10:17:48 AM
6	Store 4	Product 7	4,753.00	Ofoefule Christian	10th-Jun-2018 10:17:49 AM
7	Store 5	Product 3	9,316.00	Ekene Okeke	10th-Jun-2018 10:17:50 AM
8	Store 1	Product 5	3,217.00	Ofoefule Christian	10th-Jun-2018 10:17:51 AM
9	Store 8	Product 10	9,434.00	Ofoefule Christian	10th-Jun-2018 10:17:52 AM
10	Store 7	Product 6	4,460.00	Ofoefule Christian	10th-Jun-2018 10:17:54 AM

Output operation for MySQL_Neo4j

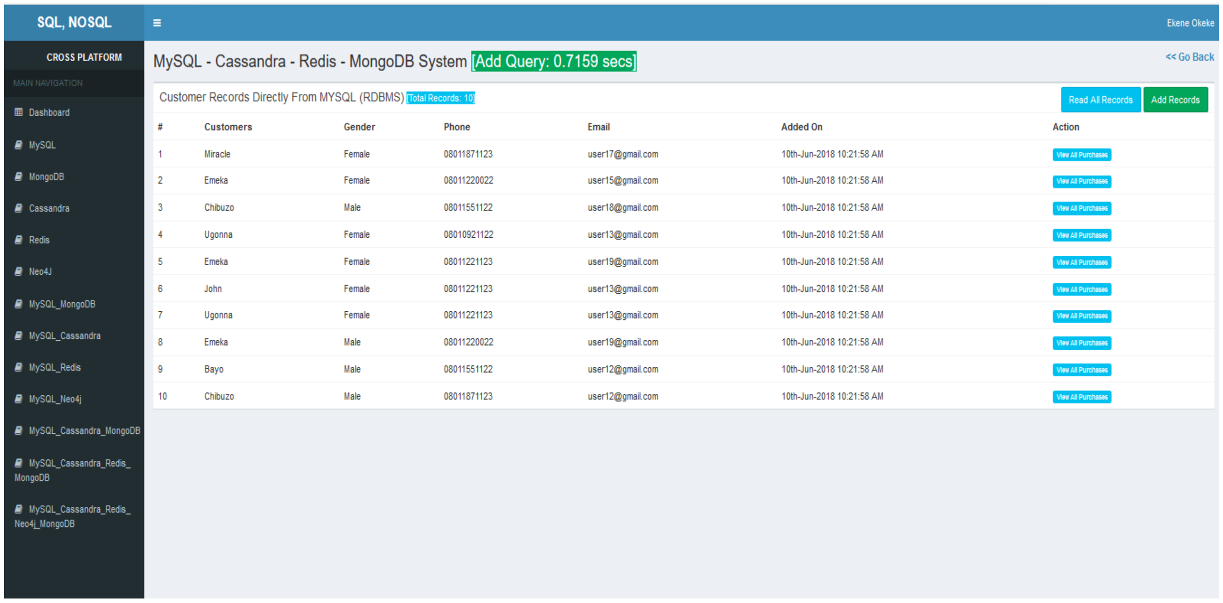


MySQL - Cassandra - MongoDB System [Add Query: 0.73379 secs]

Customer Records Directly From MYSQL (RDBMS) [Total Records: 10]

#	Customers	Gender	Phone	Email	Added On	Action
1	Adam	Male	08011551122	user14@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
2	Emeka	Female	08011551122	user15@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
3	Udoka	Male	08010921122	user14@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
4	Chibuzo	Female	08011221122	user15@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
5	Miracle	Female	08011221123	user18@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
6	Miracle	Female	08011220022	user19@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
7	Ugonna	Female	08011221123	user12@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
8	Chibuzo	Female	08011341123	user14@gmail.com	10th-Jun-2018 10:20:58 AM	View All Purchases
9	Ugonna	Male	08011221123	user16@gmail.com	10th-Jun-2018 10:20:57 AM	View All Purchases
10	Sheyi	Male	08011871123	user16@gmail.com	10th-Jun-2018 10:20:57 AM	View All Purchases

Input operation for MySQL_Cassandra_MongoDB

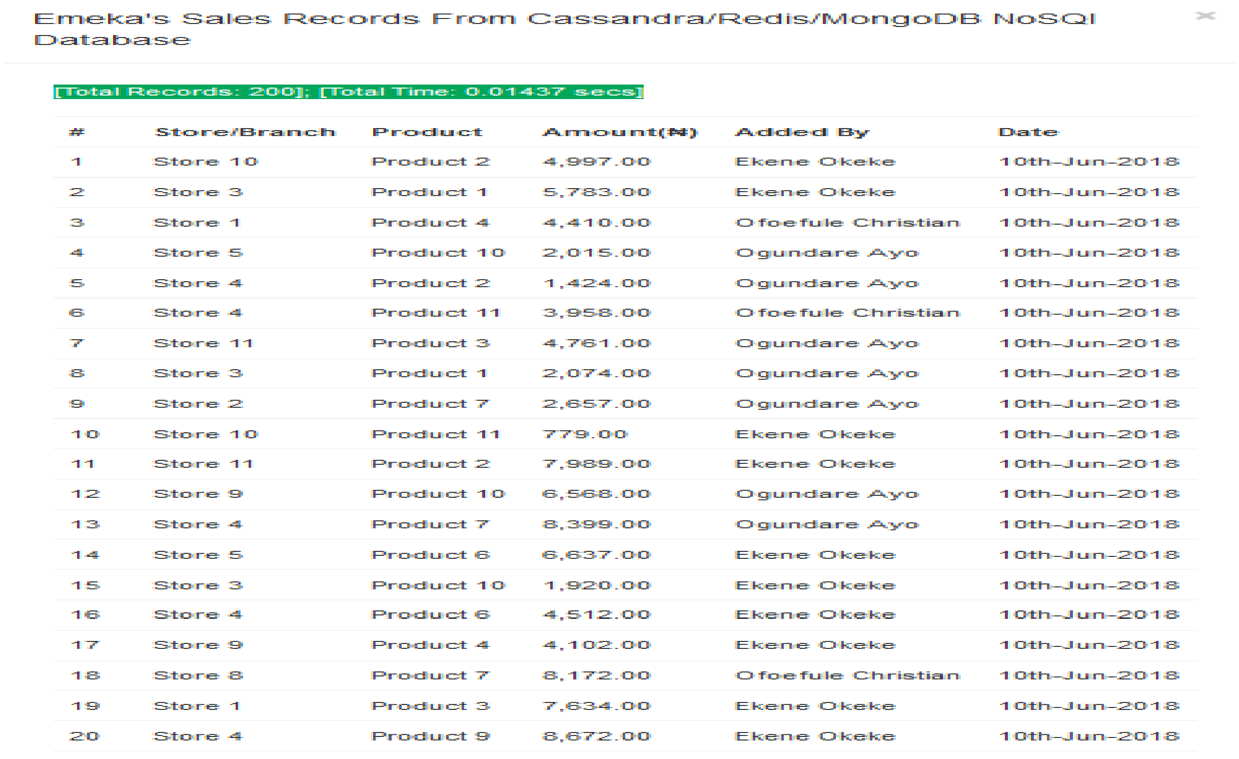


MySQL - Cassandra - Redis - MongoDB System [Add Query: 0.7159 secs]

Customer Records Directly From MYSQL (RDBMS) [Total Records: 10]

#	Customers	Gender	Phone	Email	Added On	Action
1	Miracle	Female	08011871123	user17@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
2	Emeka	Female	08011220022	user15@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
3	Chibuzo	Male	08011551122	user18@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
4	Ugonna	Female	08010921122	user13@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
5	Emeka	Female	08011221123	user19@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
6	John	Female	08011221123	user13@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
7	Ugonna	Female	08011221123	user13@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
8	Emeka	Male	08011220022	user19@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
9	Bayo	Male	08011551122	user12@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases
10	Chibuzo	Male	08011871123	user12@gmail.com	10th-Jun-2018 10:21:58 AM	View All Purchases

Input operation for MySQL_Cassandra_Redis_MongoDB

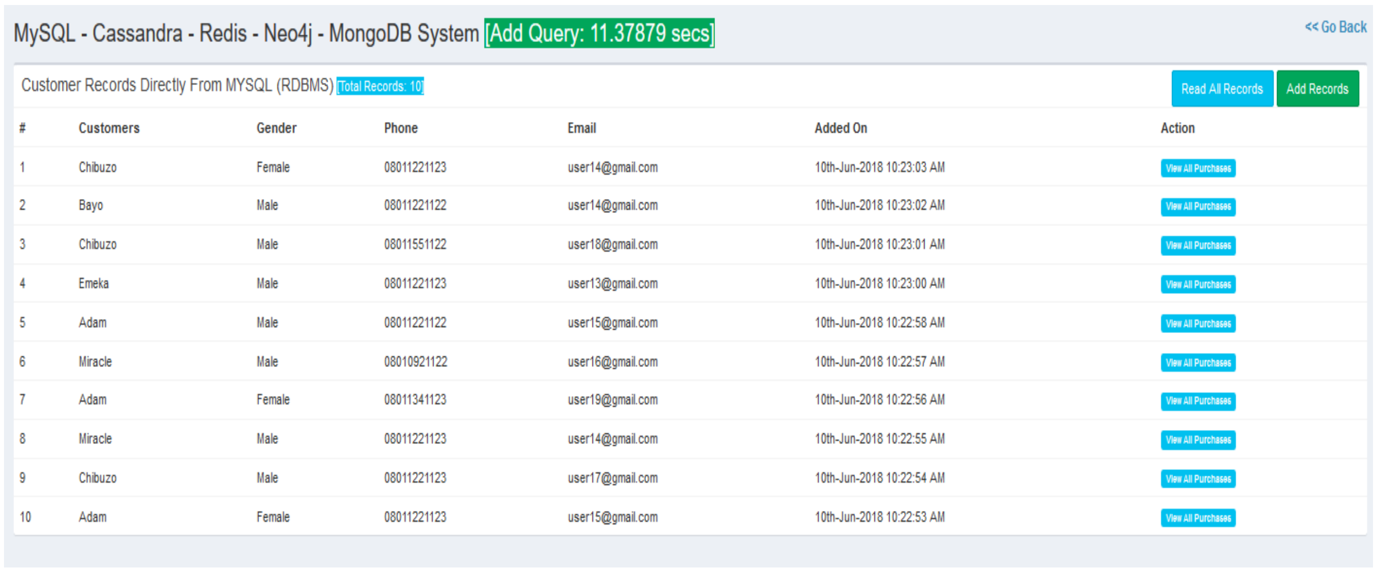


Emeka's Sales Records From Cassandra/Redis/MongoDB NoSQl Database

[Total Records: 200]; [Total Time: 0.01437 secs]

#	Store/Branch	Product	Amount(₦)	Added By	Date
1	Store 10	Product 2	4,997.00	Ekene Okeke	10th-Jun-2018
2	Store 3	Product 1	5,783.00	Ekene Okeke	10th-Jun-2018
3	Store 1	Product 4	4,410.00	Ofoefule Christian	10th-Jun-2018
4	Store 5	Product 10	2,015.00	Ogundare Ayo	10th-Jun-2018
5	Store 4	Product 2	1,424.00	Ogundare Ayo	10th-Jun-2018
6	Store 4	Product 11	3,958.00	Ofoefule Christian	10th-Jun-2018
7	Store 11	Product 3	4,761.00	Ogundare Ayo	10th-Jun-2018
8	Store 3	Product 1	2,074.00	Ogundare Ayo	10th-Jun-2018
9	Store 2	Product 7	2,657.00	Ogundare Ayo	10th-Jun-2018
10	Store 10	Product 11	779.00	Ekene Okeke	10th-Jun-2018
11	Store 11	Product 2	7,989.00	Ekene Okeke	10th-Jun-2018
12	Store 9	Product 10	6,568.00	Ogundare Ayo	10th-Jun-2018
13	Store 4	Product 7	8,399.00	Ogundare Ayo	10th-Jun-2018
14	Store 5	Product 6	6,637.00	Ekene Okeke	10th-Jun-2018
15	Store 3	Product 10	1,920.00	Ekene Okeke	10th-Jun-2018
16	Store 4	Product 6	4,512.00	Ekene Okeke	10th-Jun-2018
17	Store 9	Product 4	4,102.00	Ekene Okeke	10th-Jun-2018
18	Store 8	Product 7	8,172.00	Ofoefule Christian	10th-Jun-2018
19	Store 1	Product 3	7,634.00	Ekene Okeke	10th-Jun-2018
20	Store 4	Product 9	8,672.00	Ekene Okeke	10th-Jun-2018

Output operation for MySQL_Cassandra_Redis_MongoDB



MySQL - Cassandra - Redis - Neo4j - MongoDB System [Add Query: 11.37879 secs]

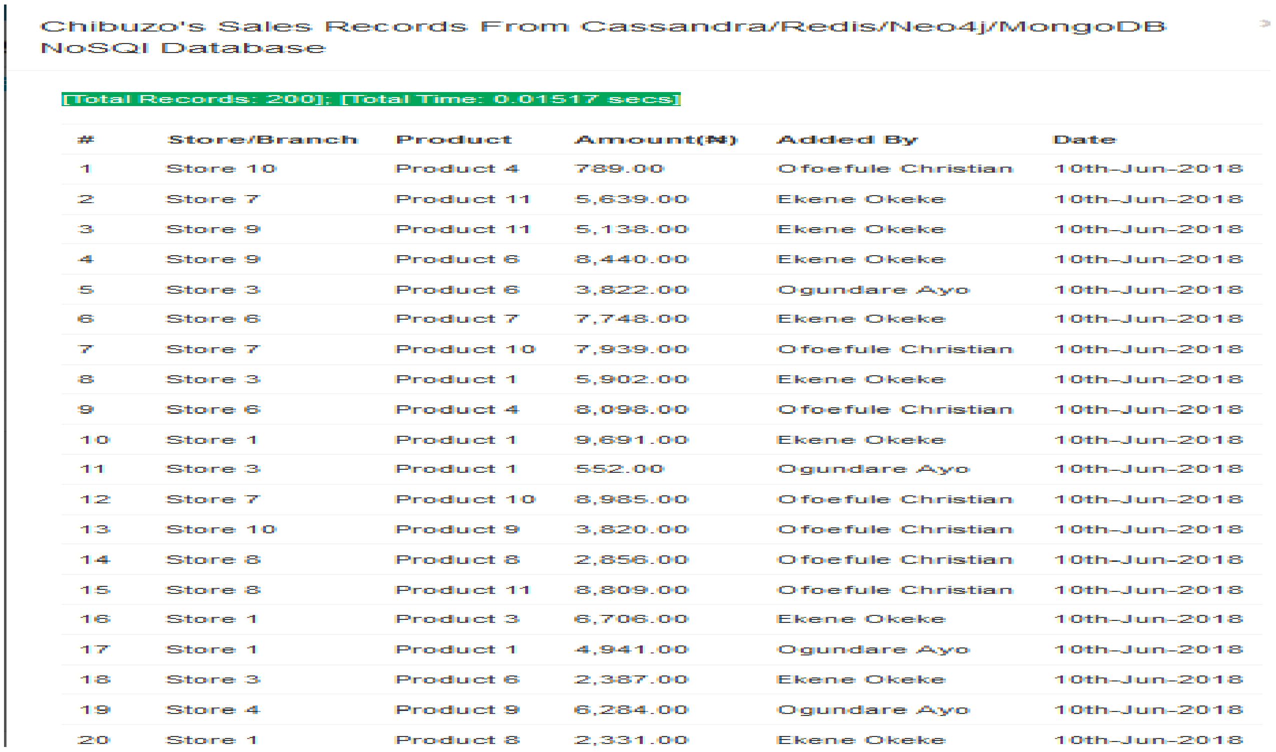
<< Go Back

Customer Records Directly From MYSQL (RDBMS) [Total Records: 10]

Read All Records | Add Records

#	Customers	Gender	Phone	Email	Added On	Action
1	Chibuzo	Female	08011221123	user14@gmail.com	10th-Jun-2018 10:23:03 AM	View All Purchases
2	Bayo	Male	08011221122	user14@gmail.com	10th-Jun-2018 10:23:02 AM	View All Purchases
3	Chibuzo	Male	08011551122	user18@gmail.com	10th-Jun-2018 10:23:01 AM	View All Purchases
4	Emeka	Male	08011221123	user13@gmail.com	10th-Jun-2018 10:23:00 AM	View All Purchases
5	Adam	Male	08011221122	user15@gmail.com	10th-Jun-2018 10:22:58 AM	View All Purchases
6	Miracle	Male	08010921122	user16@gmail.com	10th-Jun-2018 10:22:57 AM	View All Purchases
7	Adam	Female	08011341123	user19@gmail.com	10th-Jun-2018 10:22:56 AM	View All Purchases
8	Miracle	Male	08011221123	user14@gmail.com	10th-Jun-2018 10:22:55 AM	View All Purchases
9	Chibuzo	Male	08011221123	user17@gmail.com	10th-Jun-2018 10:22:54 AM	View All Purchases
10	Adam	Female	08011221123	user15@gmail.com	10th-Jun-2018 10:22:53 AM	View All Purchases

Input operation for MySQL_Cassandra_Redis_Neo4j_MongoDB



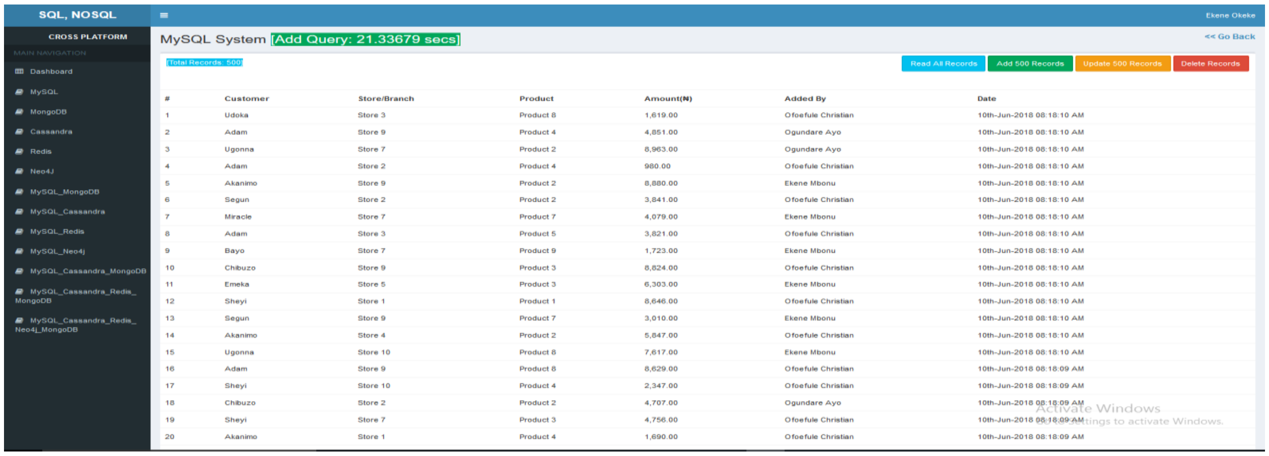
Chibuzo's Sales Records From Cassandra/Redis/Neo4j/MongoDB NoSQl Database

[Total Records: 200]; [Total Time: 0.01517 secs]

#	Store/Branch	Product	Amount(₦)	Added By	Date
1	Store 10	Product 4	789.00	Ofoefule Christian	10th-Jun-2018
2	Store 7	Product 11	5,639.00	Ekene Okeke	10th-Jun-2018
3	Store 9	Product 11	5,138.00	Ekene Okeke	10th-Jun-2018
4	Store 9	Product 6	8,440.00	Ekene Okeke	10th-Jun-2018
5	Store 3	Product 6	3,822.00	Ogundare Ayo	10th-Jun-2018
6	Store 6	Product 7	7,748.00	Ekene Okeke	10th-Jun-2018
7	Store 7	Product 10	7,939.00	Ofoefule Christian	10th-Jun-2018
8	Store 3	Product 1	5,902.00	Ekene Okeke	10th-Jun-2018
9	Store 6	Product 4	8,098.00	Ofoefule Christian	10th-Jun-2018
10	Store 1	Product 1	9,691.00	Ekene Okeke	10th-Jun-2018
11	Store 3	Product 1	552.00	Ogundare Ayo	10th-Jun-2018
12	Store 7	Product 10	8,985.00	Ofoefule Christian	10th-Jun-2018
13	Store 10	Product 9	3,820.00	Ofoefule Christian	10th-Jun-2018
14	Store 8	Product 8	2,856.00	Ofoefule Christian	10th-Jun-2018
15	Store 8	Product 11	8,809.00	Ofoefule Christian	10th-Jun-2018
16	Store 1	Product 3	6,706.00	Ekene Okeke	10th-Jun-2018
17	Store 1	Product 1	4,941.00	Ogundare Ayo	10th-Jun-2018
18	Store 3	Product 6	2,387.00	Ekene Okeke	10th-Jun-2018
19	Store 4	Product 9	6,284.00	Ogundare Ayo	10th-Jun-2018
20	Store 1	Product 8	2,331.00	Ekene Okeke	10th-Jun-2018

Output operation for MySQL_Cassandra_Redis_Neo4j_Mongo

MySQL System



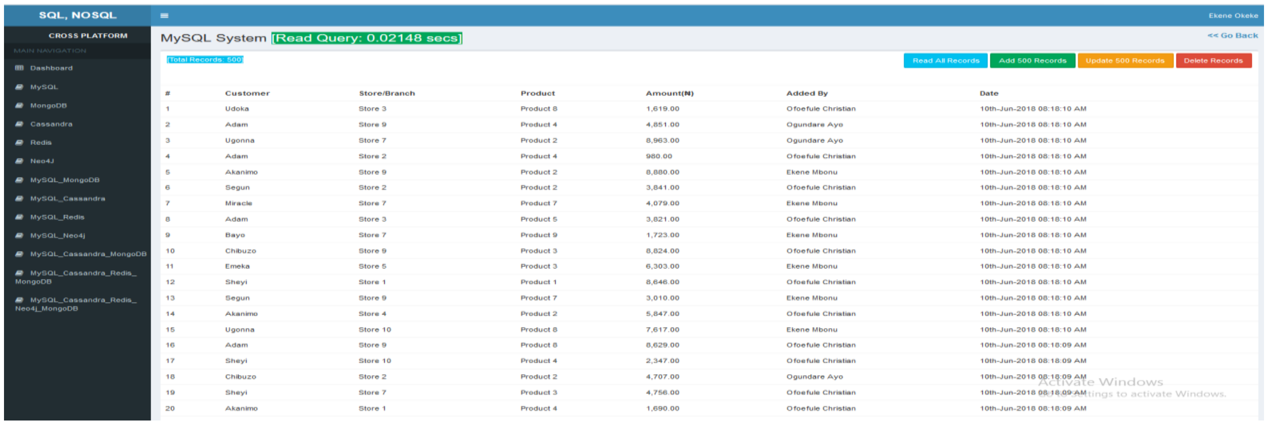
MySQL System [Add Query: 21.33679 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 3	Product 8	1,619.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
2	Adam	Store 9	Product 4	4,851.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
3	Ugonna	Store 7	Product 2	8,963.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
4	Adam	Store 2	Product 4	980.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
5	Akanimo	Store 9	Product 2	8,880.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
6	Segun	Store 2	Product 2	3,841.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
7	Miracle	Store 7	Product 7	4,079.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
8	Adam	Store 3	Product 5	3,821.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
9	Bayo	Store 7	Product 9	1,723.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
10	Chibuzo	Store 9	Product 3	8,824.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
11	Emeka	Store 5	Product 3	6,303.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
12	Sheyi	Store 1	Product 1	8,646.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
13	Segun	Store 9	Product 7	3,010.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
14	Akanimo	Store 4	Product 2	5,847.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
15	Ugonna	Store 10	Product 8	7,617.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
16	Adam	Store 9	Product 8	8,629.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
17	Sheyi	Store 10	Product 4	2,347.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
18	Chibuzo	Store 2	Product 2	4,707.00	Ogundare Ayo	10th-Jun-2018 08:18:09 AM
19	Sheyi	Store 7	Product 3	4,756.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
20	Akanimo	Store 1	Product 4	1,690.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM

MySQL Add Query for 500 records

The Insertion operation was completed in 21.34 seconds. The time displayed shows that MySQL system is slow for insertions.



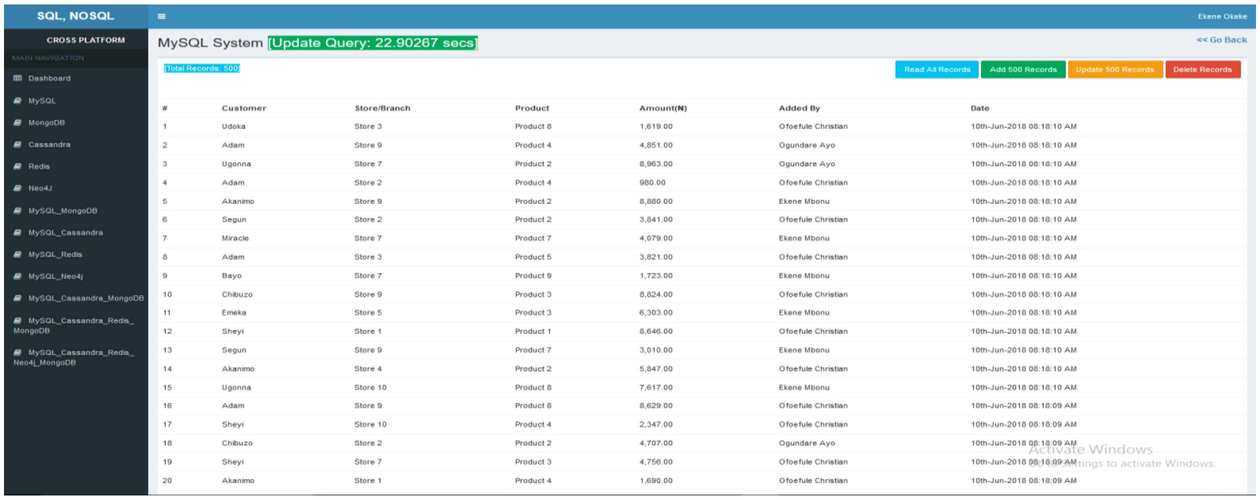
MySQL System [Read Query: 0.02148 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 3	Product 8	1,619.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
2	Adam	Store 9	Product 4	4,851.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
3	Ugonna	Store 7	Product 2	8,963.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
4	Adam	Store 2	Product 4	980.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
5	Akanimo	Store 9	Product 2	8,880.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
6	Segun	Store 2	Product 2	3,841.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
7	Miracle	Store 7	Product 7	4,079.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
8	Adam	Store 3	Product 5	3,821.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
9	Bayo	Store 7	Product 9	1,723.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
10	Chibuzo	Store 9	Product 3	8,824.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
11	Emeka	Store 5	Product 3	6,303.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
12	Sheyi	Store 1	Product 1	8,646.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
13	Segun	Store 9	Product 7	3,010.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
14	Akanimo	Store 4	Product 2	5,847.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
15	Ugonna	Store 10	Product 8	7,617.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
16	Adam	Store 9	Product 8	8,629.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
17	Sheyi	Store 10	Product 4	2,347.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
18	Chibuzo	Store 2	Product 2	4,707.00	Ogundare Ayo	10th-Jun-2018 08:18:09 AM
19	Sheyi	Store 7	Product 3	4,756.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
20	Akanimo	Store 1	Product 4	1,690.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM

MySQL Read Query for 500 records

The read operation for 500 records was completed in 0.02148 seconds. This query completion time asserts that MySQL system is good for read operations.

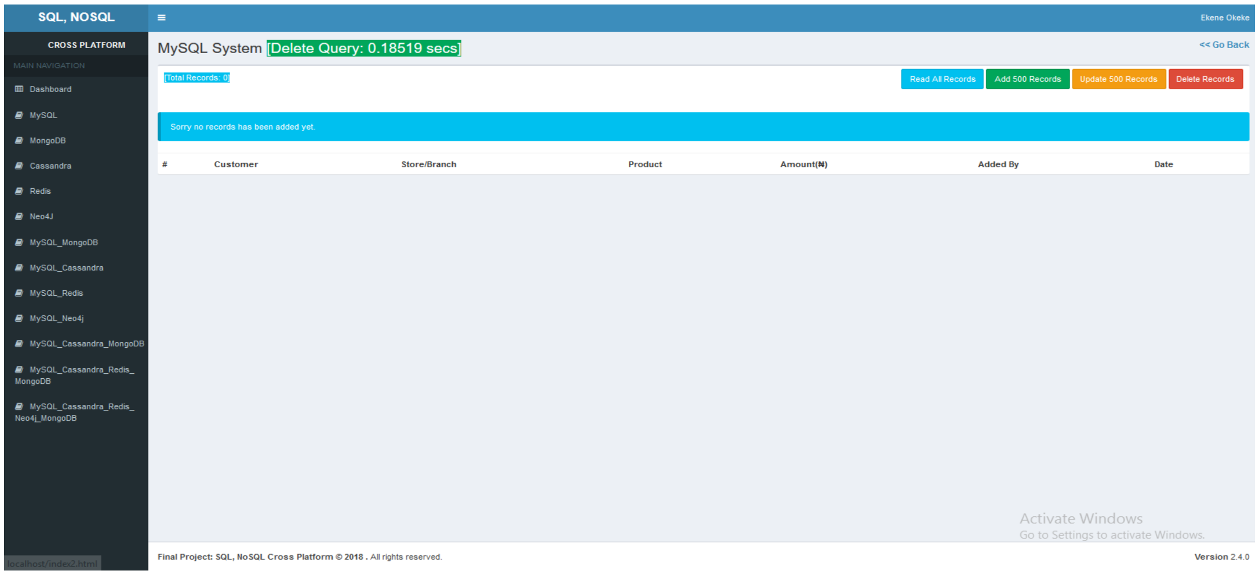

MySQL System [Update Query: 22.90267 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 3	Product 8	1,619.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
2	Adam	Store 9	Product 4	4,851.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
3	Ugonna	Store 7	Product 2	8,963.00	Ogundare Ayo	10th-Jun-2018 08:18:10 AM
4	Adam	Store 2	Product 4	980.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
5	Akanimo	Store 9	Product 2	8,880.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
6	Segun	Store 2	Product 2	3,841.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
7	Miracle	Store 7	Product 7	4,079.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
8	Adam	Store 3	Product 5	3,821.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
9	Bayo	Store 7	Product 9	1,723.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
10	Chibuzo	Store 9	Product 3	8,824.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
11	Emeka	Store 5	Product 3	6,303.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
12	Sheyi	Store 1	Product 1	8,646.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
13	Segun	Store 9	Product 7	3,010.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
14	Akanimo	Store 4	Product 2	5,847.00	Ofoefule Christian	10th-Jun-2018 08:18:10 AM
15	Ugonna	Store 10	Product 8	7,617.00	Ekene Mbonu	10th-Jun-2018 08:18:10 AM
16	Adam	Store 9	Product 8	8,629.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
17	Sheyi	Store 10	Product 4	2,347.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
18	Chibuzo	Store 2	Product 2	4,707.00	Ogundare Ayo	10th-Jun-2018 08:18:09 AM
19	Sheyi	Store 7	Product 3	4,756.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM
20	Akanimo	Store 1	Product 4	1,690.00	Ofoefule Christian	10th-Jun-2018 08:18:09 AM

Update Query for 500 records

The update query for 500 records was completed in 22.90267 seconds. The completion time for the 500 records shows that MySQL system is not suitable for update operations.


MySQL System [Delete Query: 0.18519 secs]

[Total Records: 0]

Sorry no records has been added yet.

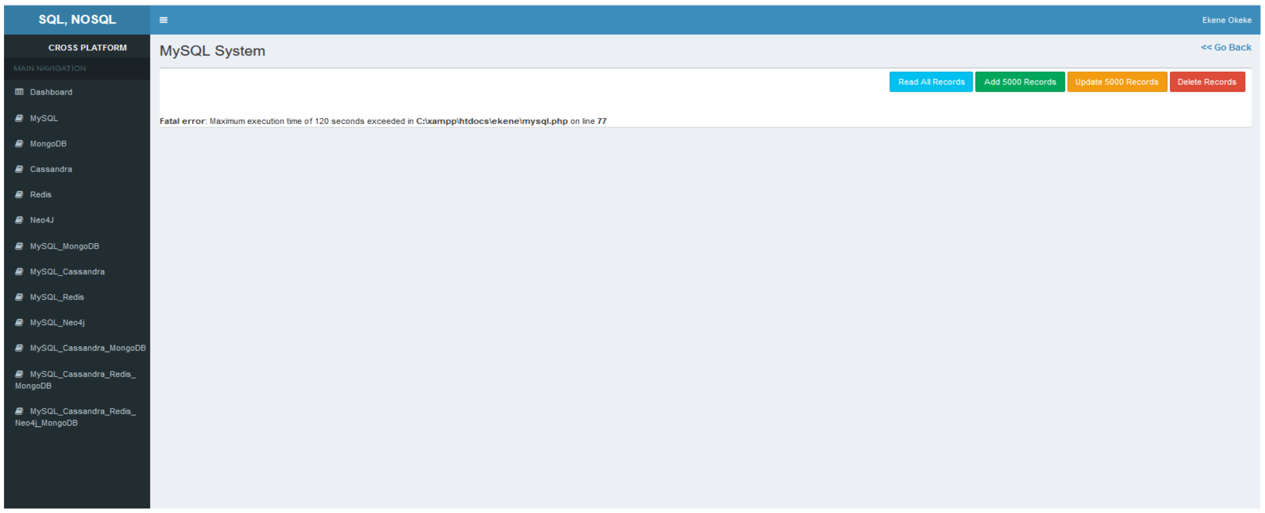
#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Final Project: SQL, NoSQL Cross Platform © 2018 . All rights reserved.

Version 2.4.0

Delete Query for 500 records

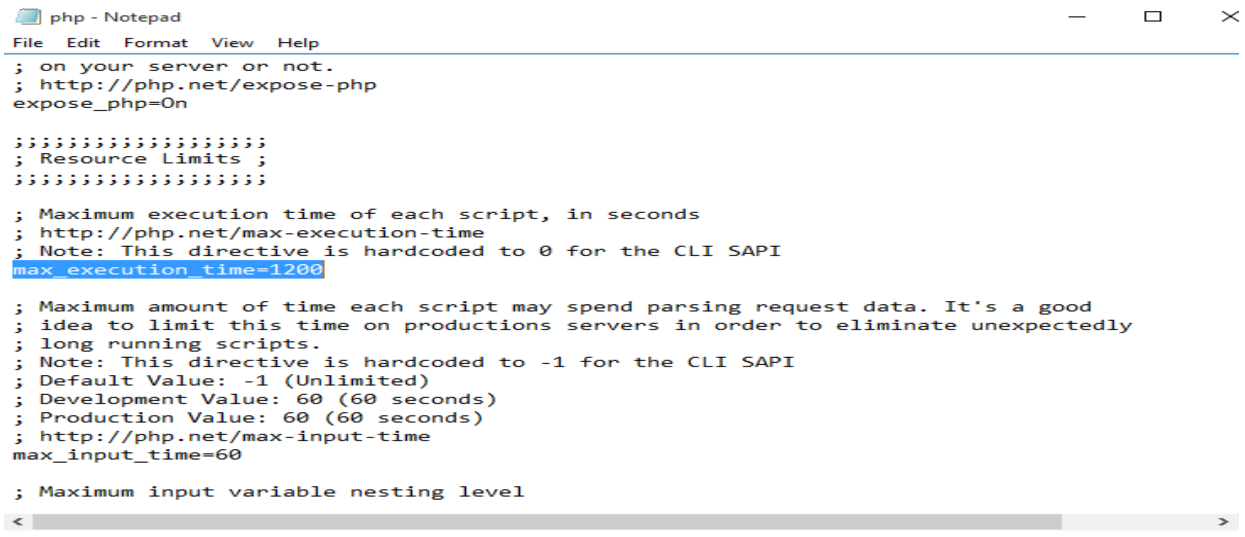
The delete operation for 500 records was completed in 0.185519 seconds. The result confirms that MySQL performs deletion fast.



```
Fatal error: Maximum execution time of 120 seconds exceeded in C:\xampp\htdocs\ekene\mysql.php on line 77
```

Excerption as a result of system timeout in MySQL

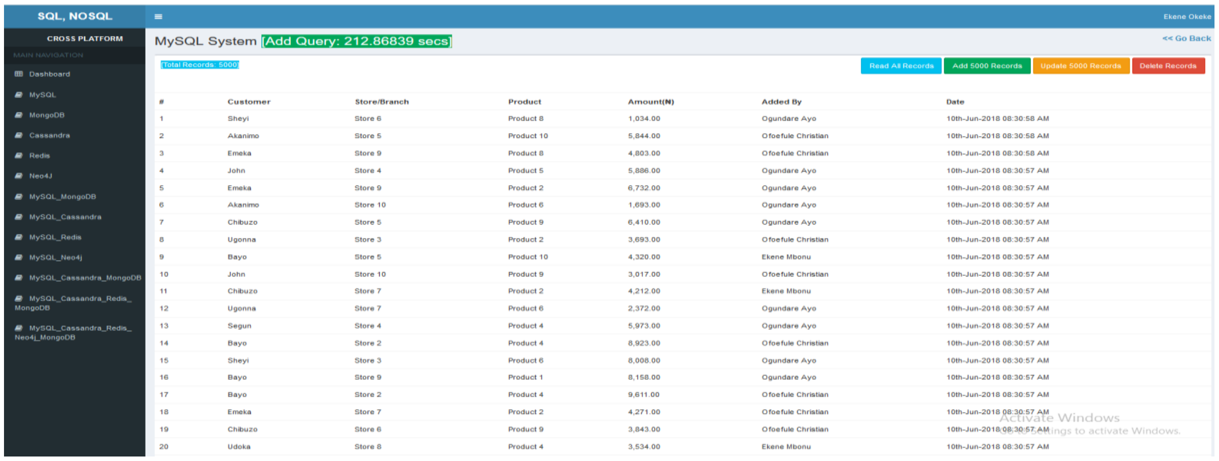
The system timed out as a result of the time limit of 120 seconds (2 minutes) which was set for running tests for the systems. The time limit was increased to 1,200 seconds as a result of the uncertainty of completion times experienced during the testing of the MySQL system.



```
; on your server or not.
; http://php.net/expose-php
expose_php=On

;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

; Maximum execution time of each script, in seconds
; http://php.net/max-execution-time
; Note: This directive is hardcoded to 0 for the CLI SAPI
max_execution_time=1200

; Maximum amount of time each script may spend parsing request data. It's a good
; idea to limit this time on productions servers in order to eliminate unexpectedly
; long running scripts.
; Note: This directive is hardcoded to -1 for the CLI SAPI
; Default Value: -1 (Unlimited)
; Development Value: 60 (60 seconds)
; Production Value: 60 (60 seconds)
; http://php.net/max-input-time
max_input_time=60

; Maximum input variable nesting level
```

Execution time increments for MySQL system

The query execution time was increased for MySQL when the insertion was increased from 500 to 5000 records. The increment was performed to decipher the MySQL performance when records are larger. The add query operation which has already been noticed to be slow for 500 records and was completed in 21.33679 seconds timed out for the addition of 5,000 records.



MySQL System [Add Query: 212.86839 secs]

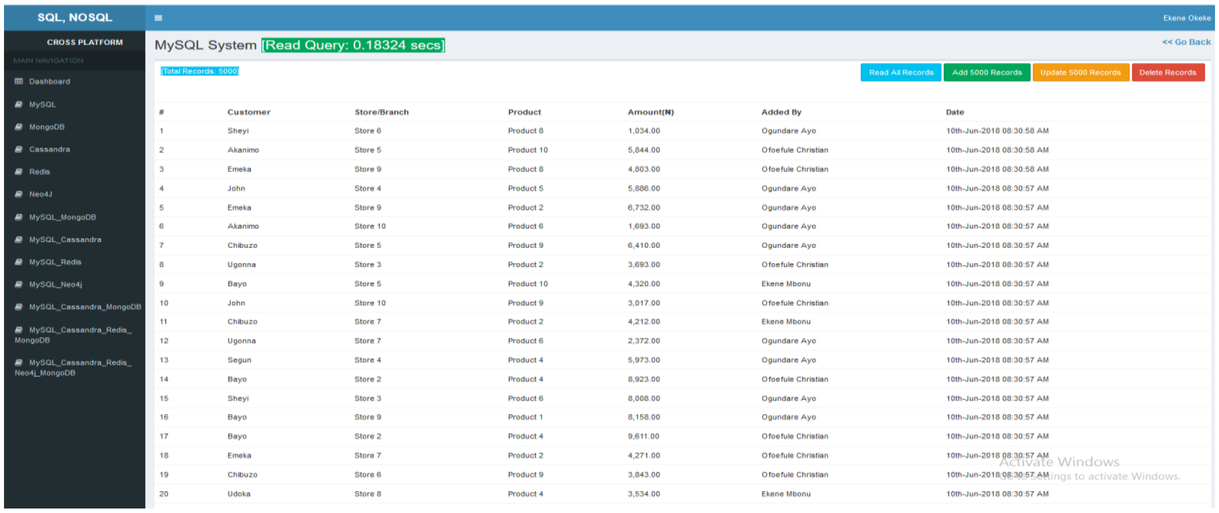
[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 6	Product 8	1,034.00	Ogundare Ayo	10th-Jun-2018 08:30:58 AM
2	Akanimo	Store 5	Product 10	5,844.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
3	Emeka	Store 9	Product 8	4,803.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
4	John	Store 4	Product 5	5,886.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
5	Emeka	Store 9	Product 2	6,732.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
6	Akanimo	Store 10	Product 6	1,693.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
7	Chibuzo	Store 5	Product 9	6,410.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
8	Ugonna	Store 3	Product 2	3,693.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
9	Bayo	Store 5	Product 10	4,320.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
10	John	Store 10	Product 9	3,017.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
11	Chibuzo	Store 7	Product 2	4,212.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
12	Ugonna	Store 7	Product 6	2,372.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
13	Segun	Store 4	Product 4	5,973.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
14	Bayo	Store 2	Product 4	8,923.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
15	Sheyi	Store 3	Product 6	8,008.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
16	Bayo	Store 9	Product 1	8,158.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
17	Bayo	Store 2	Product 4	9,611.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
18	Emeka	Store 7	Product 2	4,271.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
19	Chibuzo	Store 6	Product 9	3,843.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
20	Udoka	Store 8	Product 4	3,534.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM

Activate Windows
Go to Settings to activate Windows.

MySQL Add Query for 5,000 records

Insertion of 5,000 records in MySQL system was completed in 212.87 seconds which is approximately 4 minutes. As has been deduced from the insertion of 500 records which was completed in 21.34 seconds, MySQL system is not a good fit for performing insertions.



MySQL System [Read Query: 0.18324 secs]

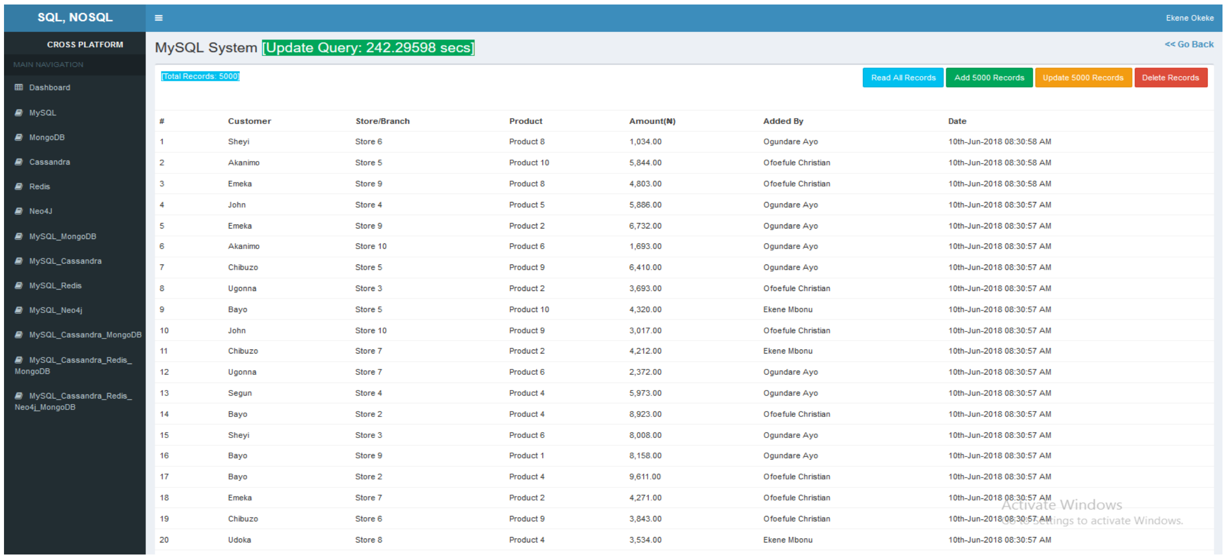
[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 6	Product 8	1,034.00	Ogundare Ayo	10th-Jun-2018 08:30:58 AM
2	Akanimo	Store 5	Product 10	5,844.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
3	Emeka	Store 9	Product 8	4,803.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
4	John	Store 4	Product 5	5,886.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
5	Emeka	Store 9	Product 2	6,732.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
6	Akanimo	Store 10	Product 6	1,693.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
7	Chibuzo	Store 5	Product 9	6,410.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
8	Ugonna	Store 3	Product 2	3,693.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
9	Bayo	Store 5	Product 10	4,320.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
10	John	Store 10	Product 9	3,017.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
11	Chibuzo	Store 7	Product 2	4,212.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
12	Ugonna	Store 7	Product 6	2,372.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
13	Segun	Store 4	Product 4	5,973.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
14	Bayo	Store 2	Product 4	8,923.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
15	Sheyi	Store 3	Product 6	8,008.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
16	Bayo	Store 9	Product 1	8,158.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
17	Bayo	Store 2	Product 4	9,611.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
18	Emeka	Store 7	Product 2	4,271.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
19	Chibuzo	Store 6	Product 9	3,843.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
20	Udoka	Store 8	Product 4	3,534.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM

Activate Windows
Go to Settings to activate Windows.

MySQL Read Query for 5,000 records

The 5,000 records were read in 0.18324 seconds in MySQL system. This depicts strength in read operations by MySQL system.



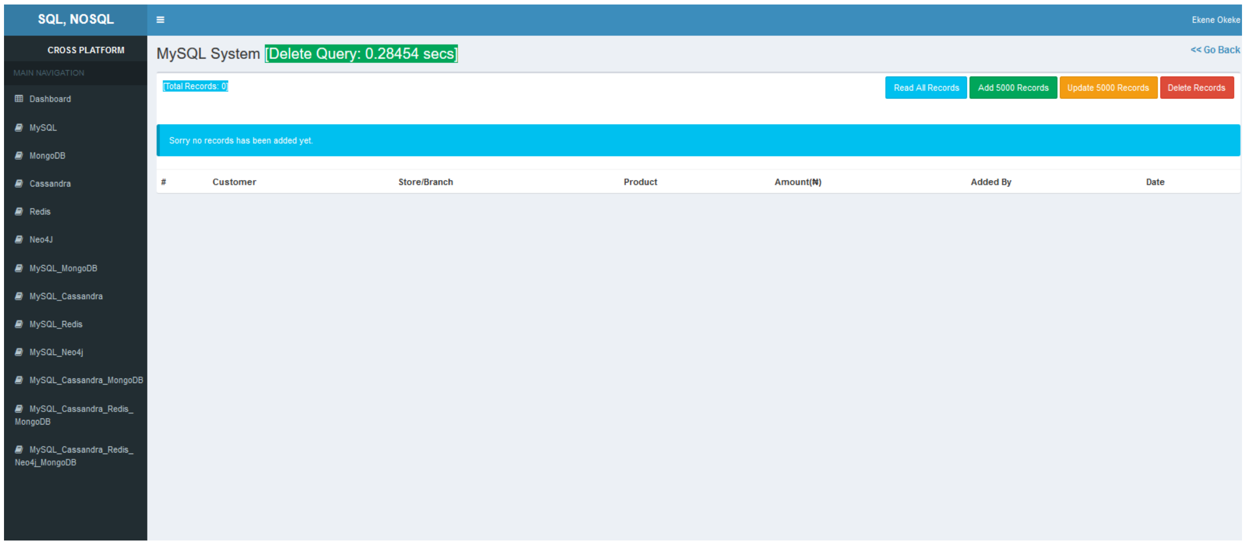
MySQL System [Update Query: 242.29598 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 6	Product 8	1,034.00	Ogundare Ayo	10th-Jun-2018 08:30:58 AM
2	Akanimo	Store 5	Product 10	5,844.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
3	Emeka	Store 9	Product 8	4,803.00	Ofoefule Christian	10th-Jun-2018 08:30:58 AM
4	John	Store 4	Product 5	5,886.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
5	Emeka	Store 9	Product 2	6,732.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
6	Akanimo	Store 10	Product 6	1,693.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
7	Chibuzo	Store 5	Product 9	6,410.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
8	Ugonna	Store 3	Product 2	3,693.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
9	Bayo	Store 5	Product 10	4,320.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
10	John	Store 10	Product 9	3,017.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
11	Chibuzo	Store 7	Product 2	4,212.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM
12	Ugonna	Store 7	Product 6	2,372.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
13	Segun	Store 4	Product 4	5,973.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
14	Bayo	Store 2	Product 4	8,923.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
15	Sheyi	Store 3	Product 6	8,008.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
16	Bayo	Store 9	Product 1	8,158.00	Ogundare Ayo	10th-Jun-2018 08:30:57 AM
17	Bayo	Store 2	Product 4	9,611.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
18	Emeka	Store 7	Product 2	4,271.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
19	Chibuzo	Store 6	Product 9	3,843.00	Ofoefule Christian	10th-Jun-2018 08:30:57 AM
20	Udoka	Store 8	Product 4	3,534.00	Ekene Mbonu	10th-Jun-2018 08:30:57 AM

MySQL Update Query for 5,000 records

The query update for 5,000 records was completed in 242.30 seconds (4 minutes) in MySQL database. MySQL performs update slowly.



MySQL System [Delete Query: 0.28454 secs]

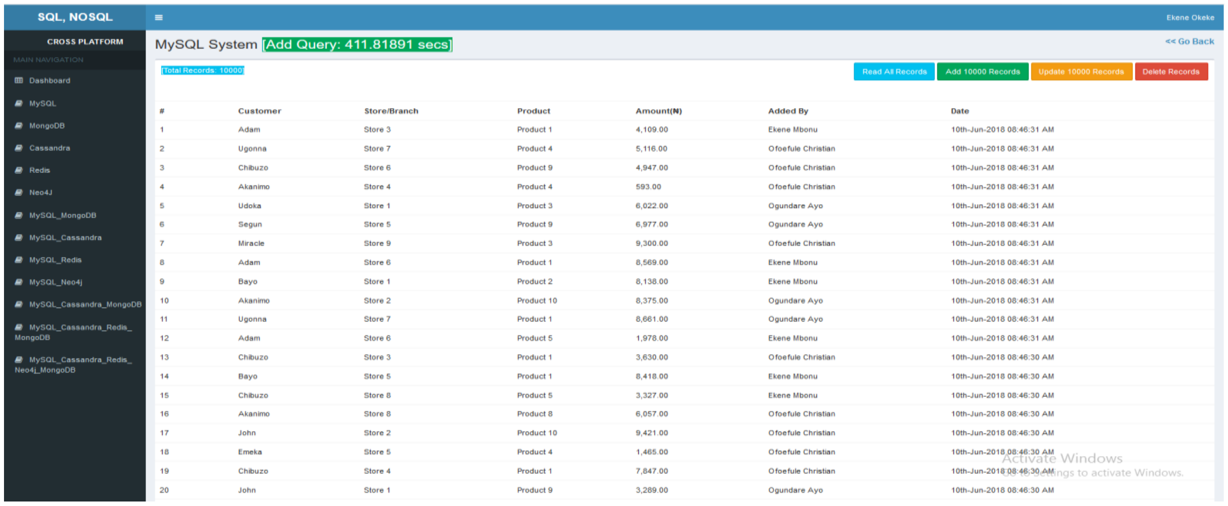
[Total Records: 0]

Sorry no records has been added yet.

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

MySQL Delete Query for 5,000 records

Deletion of 5,000 records was completed within 0.28454 seconds which shows that MySQL performs deletion fast.



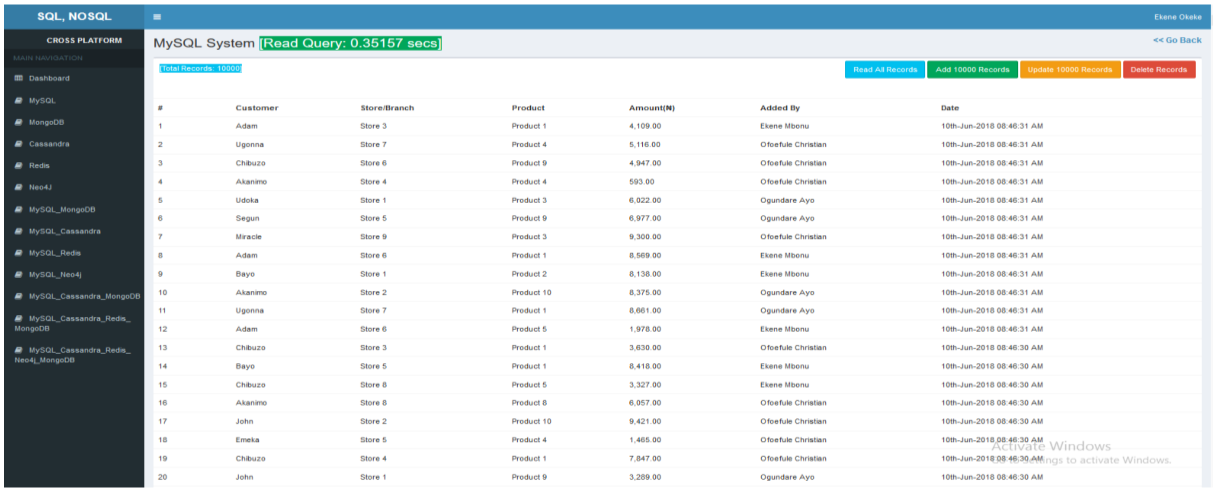
MySQL System [Add Query: 411.81891 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 3	Product 1	4,109.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
2	Ugonna	Store 7	Product 4	5,116.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
3	Chibuzo	Store 6	Product 9	4,947.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
4	Akanimo	Store 4	Product 4	593.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
5	Udoka	Store 1	Product 3	6,022.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
6	Segun	Store 5	Product 9	6,977.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
7	Miracle	Store 9	Product 3	9,300.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
8	Adam	Store 6	Product 1	8,569.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
9	Bayo	Store 1	Product 2	8,138.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
10	Akanimo	Store 2	Product 10	8,375.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
11	Ugonna	Store 7	Product 1	8,661.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
12	Adam	Store 6	Product 5	1,978.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
13	Chibuzo	Store 3	Product 1	3,630.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
14	Bayo	Store 5	Product 1	8,418.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
15	Chibuzo	Store 8	Product 5	3,327.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
16	Akanimo	Store 8	Product 8	6,057.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
17	John	Store 2	Product 10	9,421.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
18	Emeka	Store 5	Product 4	1,465.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
19	Chibuzo	Store 4	Product 1	7,847.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
20	John	Store 1	Product 9	3,289.00	Ogundare Ayo	10th-Jun-2018 08:46:30 AM

MySQL Insert operations for 10,000 records

This test shows creation or addition/insertion of 10,000 records made to MySQL system which took 411.82 seconds, approximately 7 minutes. This means that MySQL system is very slow when it comes to insertions.



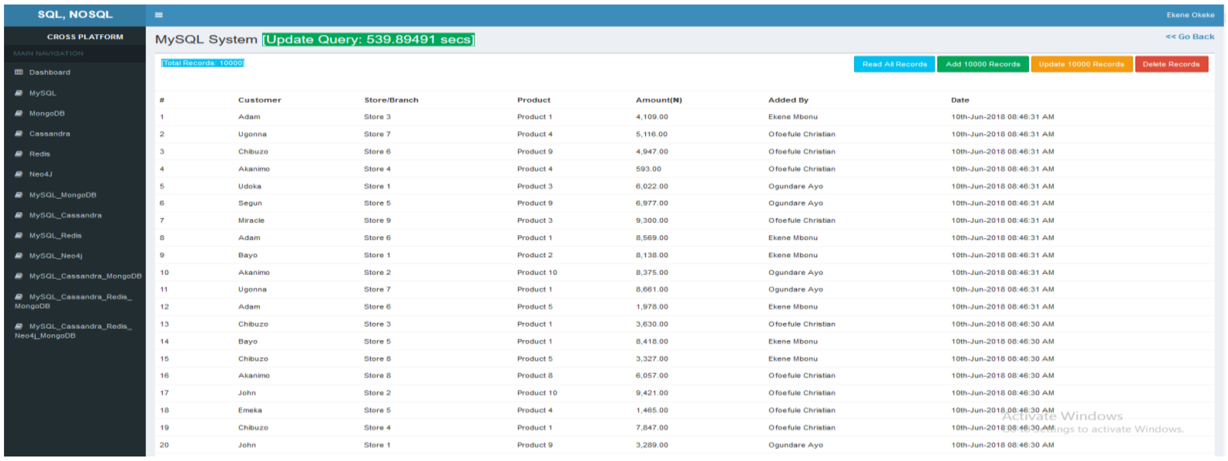
MySQL System [Read Query: 0.35157 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 3	Product 1	4,109.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
2	Ugonna	Store 7	Product 4	5,116.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
3	Chibuzo	Store 6	Product 9	4,947.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
4	Akanimo	Store 4	Product 4	593.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
5	Udoka	Store 1	Product 3	6,022.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
6	Segun	Store 5	Product 9	6,977.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
7	Miracle	Store 9	Product 3	9,300.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
8	Adam	Store 6	Product 1	8,569.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
9	Bayo	Store 1	Product 2	8,138.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
10	Akanimo	Store 2	Product 10	8,375.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
11	Ugonna	Store 7	Product 1	8,661.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
12	Adam	Store 6	Product 5	1,978.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
13	Chibuzo	Store 3	Product 1	3,630.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
14	Bayo	Store 5	Product 1	8,418.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
15	Chibuzo	Store 8	Product 5	3,327.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
16	Akanimo	Store 8	Product 8	6,057.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
17	John	Store 2	Product 10	9,421.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
18	Emeka	Store 5	Product 4	1,465.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
19	Chibuzo	Store 4	Product 1	7,847.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
20	John	Store 1	Product 9	3,289.00	Ogundare Ayo	10th-Jun-2018 08:46:30 AM

MySQL Read Query for 10,000 records

MySQL performs very fast read operations as has been shown in the figure above where 10,000 records were updated within 0.35157 seconds.



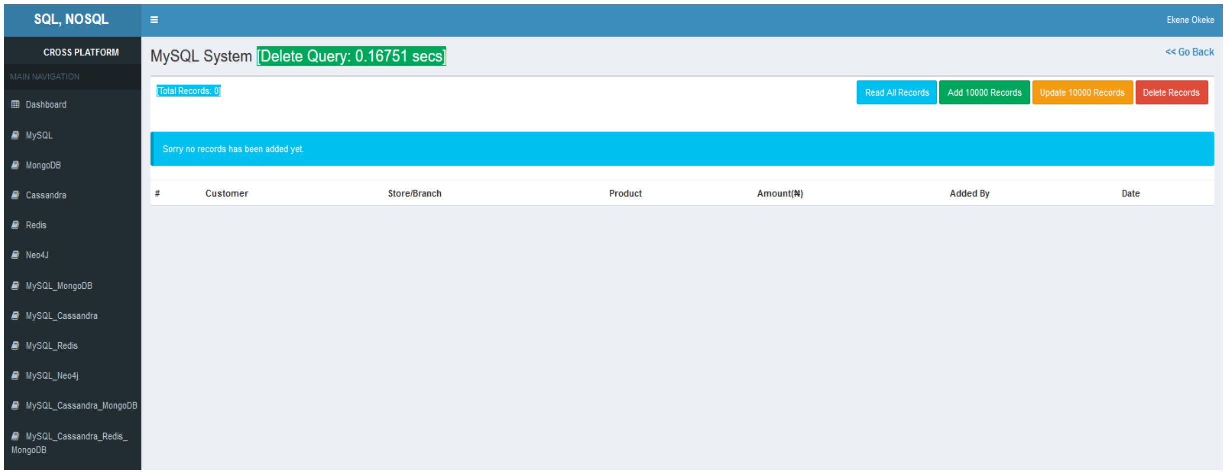
MySQL System [Update Query: 539.89491 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 3	Product 1	4,109.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
2	Ugonna	Store 7	Product 4	5,116.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
3	Chibuzo	Store 6	Product 9	4,947.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
4	Akanimo	Store 4	Product 4	593.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
5	Udoka	Store 1	Product 3	6,022.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
6	Segun	Store 5	Product 9	6,977.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
7	Miracle	Store 9	Product 3	9,300.00	Ofoefule Christian	10th-Jun-2018 08:46:31 AM
8	Adam	Store 6	Product 1	8,569.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
9	Bayo	Store 1	Product 2	8,138.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
10	Akanimo	Store 2	Product 10	8,375.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
11	Ugonna	Store 7	Product 1	8,661.00	Ogundare Ayo	10th-Jun-2018 08:46:31 AM
12	Adam	Store 6	Product 5	1,978.00	Ekene Mbonu	10th-Jun-2018 08:46:31 AM
13	Chibuzo	Store 3	Product 1	3,630.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
14	Bayo	Store 5	Product 1	8,418.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
15	Chibuzo	Store 8	Product 5	3,327.00	Ekene Mbonu	10th-Jun-2018 08:46:30 AM
16	Akanimo	Store 8	Product 8	6,057.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
17	John	Store 2	Product 10	9,421.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
18	Emeka	Store 5	Product 4	1,465.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
19	Chibuzo	Store 4	Product 1	7,847.00	Ofoefule Christian	10th-Jun-2018 08:46:30 AM
20	John	Store 1	Product 9	3,289.00	Ogundare Ayo	10th-Jun-2018 08:46:30 AM

MySQL Update Query for 10,000 records.

MySQL performed update on 10,000 records in 539.895 seconds which is 8.99 minutes (approximately 9 minutes). This test shows that MySQL is not proficient for large records updates and as such, should not be used in performing update-intensive operations.



MySQL System [Delete Query: 0.16751 secs]

[Total Records: 0]

Sorry no records has been added yet.

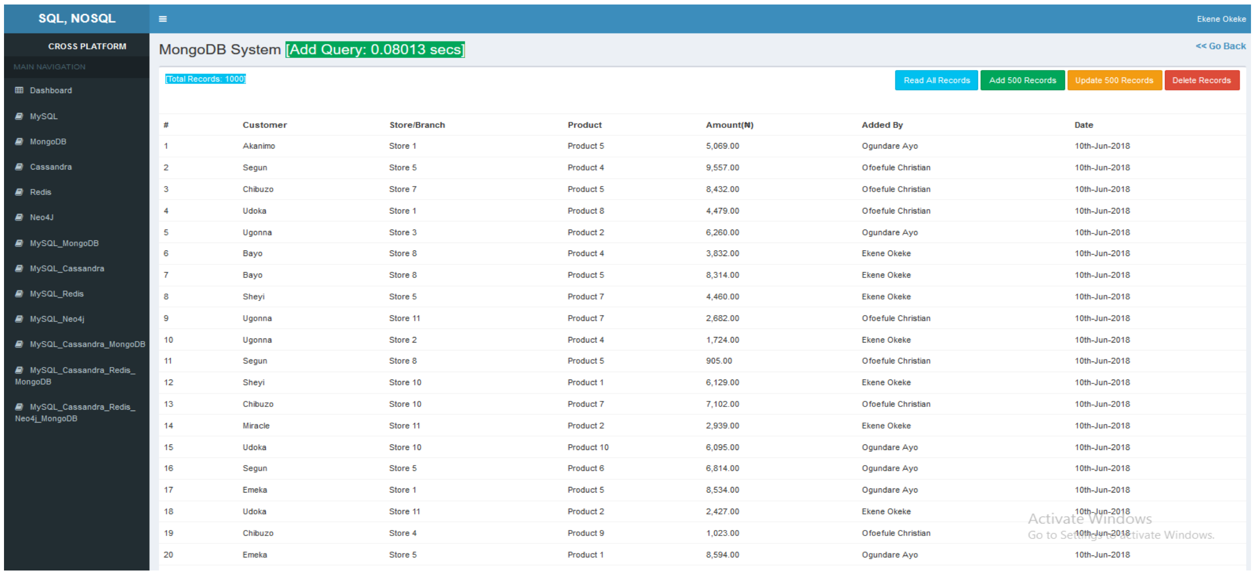
#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

MySQL Delete Query for 10,000 records

Deletion of 10,000 records was concluded in 0.168 seconds which signifies that delete operations are fast in MySQL RDBMS.

MongoDB System

The review chapter confirmed MongoDB to be read-intensive and Cassandra write-intensive. The tests below will be used as verification for that assertion.



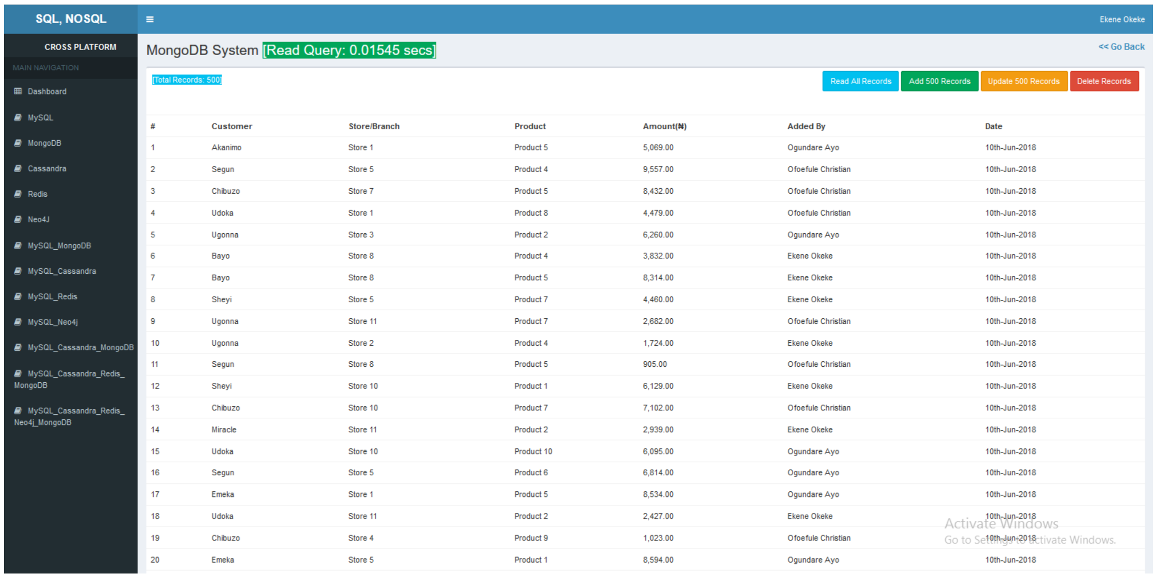
MongoDB System [Add Query: 0.08013 secs]

[Total Records: 1000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Akanimo	Store 1	Product 5	5,069.00	Ogundare Ayo	10th-Jun-2018
2	Segun	Store 5	Product 4	9,557.00	Ofoefule Christian	10th-Jun-2018
3	Chibuzo	Store 7	Product 5	8,432.00	Ofoefule Christian	10th-Jun-2018
4	Udoka	Store 1	Product 8	4,479.00	Ofoefule Christian	10th-Jun-2018
5	Ugonna	Store 3	Product 2	6,260.00	Ogundare Ayo	10th-Jun-2018
6	Bayo	Store 8	Product 4	3,832.00	Ekene Okeke	10th-Jun-2018
7	Bayo	Store 8	Product 5	8,314.00	Ekene Okeke	10th-Jun-2018
8	Sheyi	Store 5	Product 7	4,460.00	Ekene Okeke	10th-Jun-2018
9	Ugonna	Store 11	Product 7	2,682.00	Ofoefule Christian	10th-Jun-2018
10	Ugonna	Store 2	Product 4	1,724.00	Ekene Okeke	10th-Jun-2018
11	Segun	Store 8	Product 5	905.00	Ofoefule Christian	10th-Jun-2018
12	Sheyi	Store 10	Product 1	6,129.00	Ekene Okeke	10th-Jun-2018
13	Chibuzo	Store 10	Product 7	7,102.00	Ofoefule Christian	10th-Jun-2018
14	Miracle	Store 11	Product 2	2,939.00	Ekene Okeke	10th-Jun-2018
15	Udoka	Store 10	Product 10	6,095.00	Ogundare Ayo	10th-Jun-2018
16	Segun	Store 5	Product 6	6,814.00	Ogundare Ayo	10th-Jun-2018
17	Emeka	Store 1	Product 5	8,534.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 11	Product 2	2,427.00	Ekene Okeke	10th-Jun-2018
19	Chibuzo	Store 4	Product 9	1,023.00	Ofoefule Christian	10th-Jun-2018
20	Emeka	Store 5	Product 1	8,594.00	Ogundare Ayo	10th-Jun-2018

MongoDB Insert Query for 500 records

MongoDB insert Query for 500 records was performed in 0.08 seconds. 500 records were used to ascertain performance of MongoDB on relatively small record. The result shows that insert operation is fast in MongoDB.



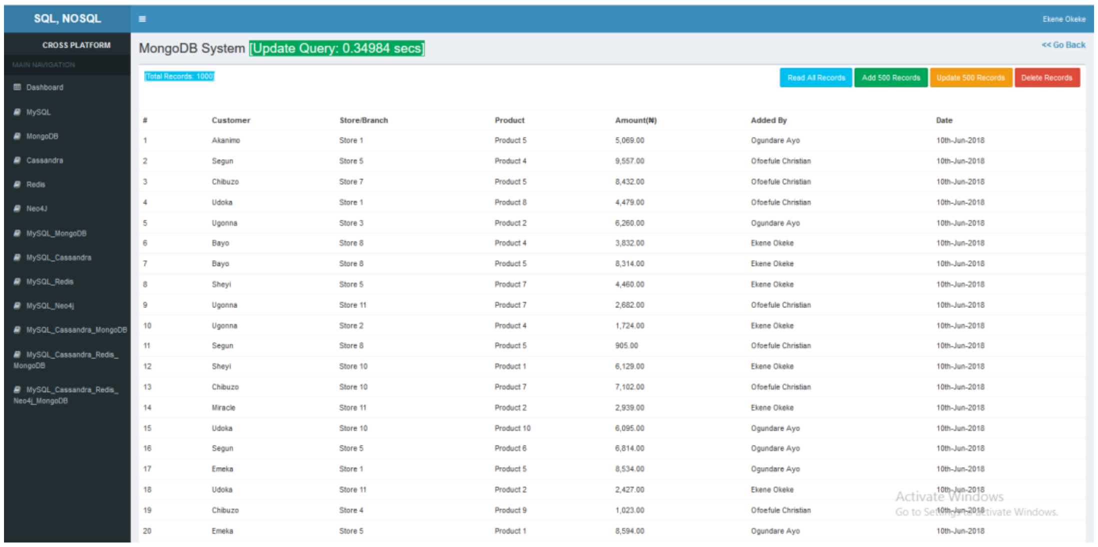
MongoDB System [Read Query: 0.01545 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Akanimo	Store 1	Product 5	5,069.00	Ogundare Ayo	10th-Jun-2018
2	Segun	Store 5	Product 4	9,557.00	Ofoefule Christian	10th-Jun-2018
3	Chibuzo	Store 7	Product 5	8,432.00	Ofoefule Christian	10th-Jun-2018
4	Udoka	Store 1	Product 8	4,479.00	Ofoefule Christian	10th-Jun-2018
5	Ugonna	Store 3	Product 2	6,260.00	Ogundare Ayo	10th-Jun-2018
6	Bayo	Store 8	Product 4	3,832.00	Ekene Okeke	10th-Jun-2018
7	Bayo	Store 8	Product 5	8,314.00	Ekene Okeke	10th-Jun-2018
8	Sheyi	Store 5	Product 7	4,460.00	Ekene Okeke	10th-Jun-2018
9	Ugonna	Store 11	Product 7	2,682.00	Ofoefule Christian	10th-Jun-2018
10	Ugonna	Store 2	Product 4	1,724.00	Ekene Okeke	10th-Jun-2018
11	Segun	Store 8	Product 5	905.00	Ofoefule Christian	10th-Jun-2018
12	Sheyi	Store 10	Product 1	6,129.00	Ekene Okeke	10th-Jun-2018
13	Chibuzo	Store 10	Product 7	7,102.00	Ofoefule Christian	10th-Jun-2018
14	Miracle	Store 11	Product 2	2,939.00	Ekene Okeke	10th-Jun-2018
15	Udoka	Store 10	Product 10	6,095.00	Ogundare Ayo	10th-Jun-2018
16	Segun	Store 5	Product 6	6,814.00	Ogundare Ayo	10th-Jun-2018
17	Emeka	Store 1	Product 5	8,534.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 11	Product 2	2,427.00	Ekene Okeke	10th-Jun-2018
19	Chibuzo	Store 4	Product 9	1,023.00	Ofoefule Christian	10th-Jun-2018
20	Emeka	Store 5	Product 1	8,594.00	Ogundare Ayo	10th-Jun-2018

MongoDB Read Query for 500 records

Read Query for 500 records in MongoDB completed in 0.01545, a record time showing speed for read operations in the MongoDB.



MongoDB System [Update Query: 0.34984 secs]

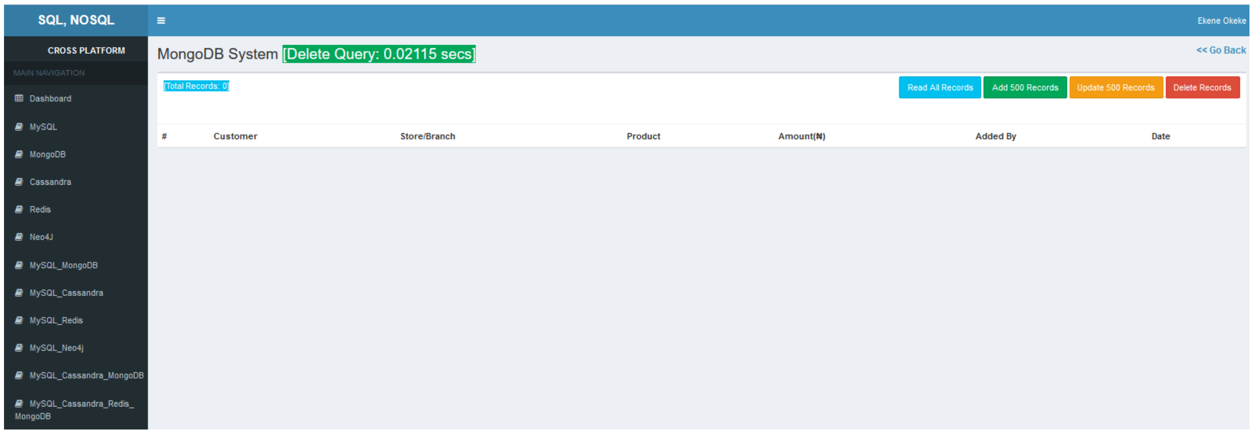
[Total Records: 1000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Akanimo	Store 1	Product 5	5,069.00	Ogundare Ayo	10th-Jun-2018
2	Segun	Store 5	Product 4	9,557.00	Ofoefule Christian	10th-Jun-2018
3	Chibuzo	Store 7	Product 5	8,432.00	Ofoefule Christian	10th-Jun-2018
4	Udoka	Store 1	Product 8	4,479.00	Ofoefule Christian	10th-Jun-2018
5	Ugonna	Store 3	Product 2	6,260.00	Ogundare Ayo	10th-Jun-2018
6	Bayo	Store 8	Product 4	3,832.00	Ekene Okeke	10th-Jun-2018
7	Bayo	Store 8	Product 5	8,314.00	Ekene Okeke	10th-Jun-2018
8	Sheyi	Store 5	Product 7	4,460.00	Ekene Okeke	10th-Jun-2018
9	Ugonna	Store 11	Product 7	2,682.00	Ofoefule Christian	10th-Jun-2018
10	Ugonna	Store 2	Product 4	1,724.00	Ekene Okeke	10th-Jun-2018
11	Segun	Store 8	Product 5	905.00	Ofoefule Christian	10th-Jun-2018
12	Sheyi	Store 10	Product 1	6,129.00	Ekene Okeke	10th-Jun-2018
13	Chibuzo	Store 10	Product 7	7,102.00	Ofoefule Christian	10th-Jun-2018
14	Miracle	Store 11	Product 2	2,939.00	Ekene Okeke	10th-Jun-2018
15	Udoka	Store 10	Product 10	6,095.00	Ogundare Ayo	10th-Jun-2018
16	Segun	Store 5	Product 6	6,814.00	Ogundare Ayo	10th-Jun-2018
17	Emeka	Store 1	Product 5	8,534.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 11	Product 2	2,427.00	Ekene Okeke	10th-Jun-2018
19	Chibuzo	Store 4	Product 9	1,023.00	Ofoefule Christian	10th-Jun-2018
20	Emeka	Store 5	Product 1	8,594.00	Ogundare Ayo	10th-Jun-2018

Activate Windows
Go to Settings to activate Windows.

MongoDB Update Query 500 records

Update query for 500 records completed in 0.35 seconds in MongoDB which shows relative speed for updates in the MongoDB NoSQL DBMS.



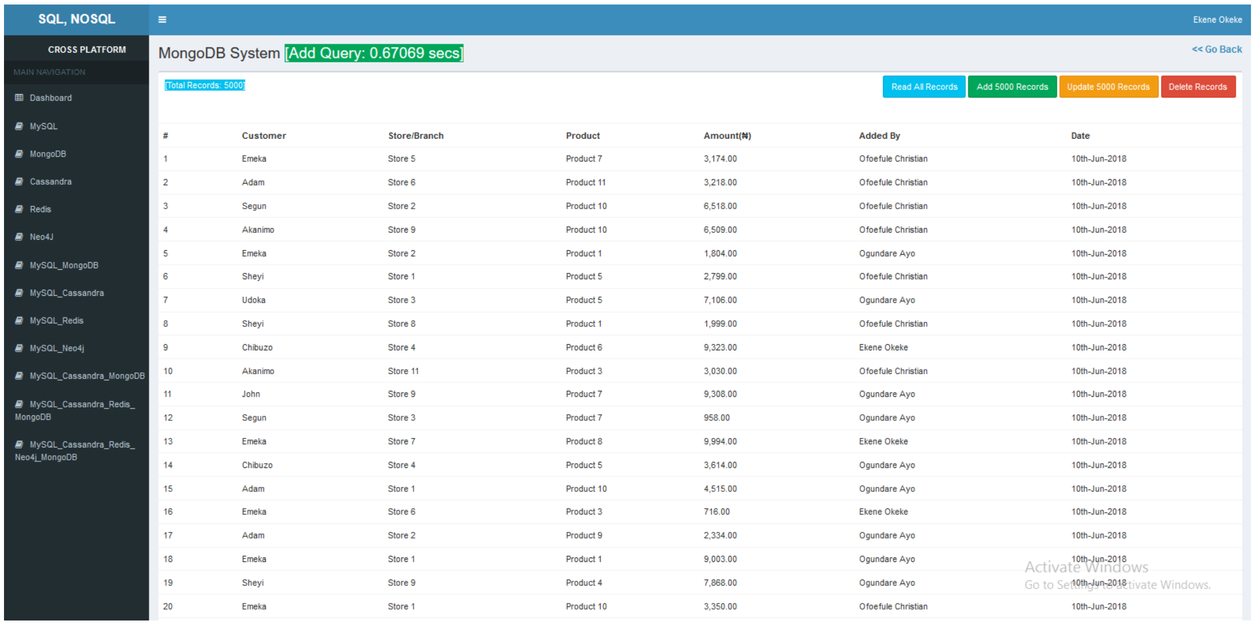
MongoDB System [Delete Query: 0.02115 secs]

[Total Records: 0]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

MongoDB Delete Query for 500 records

Delete query for MongoDB system completed in 0.02115 seconds in MongoDB proves that MongoDB performs fast deletion.



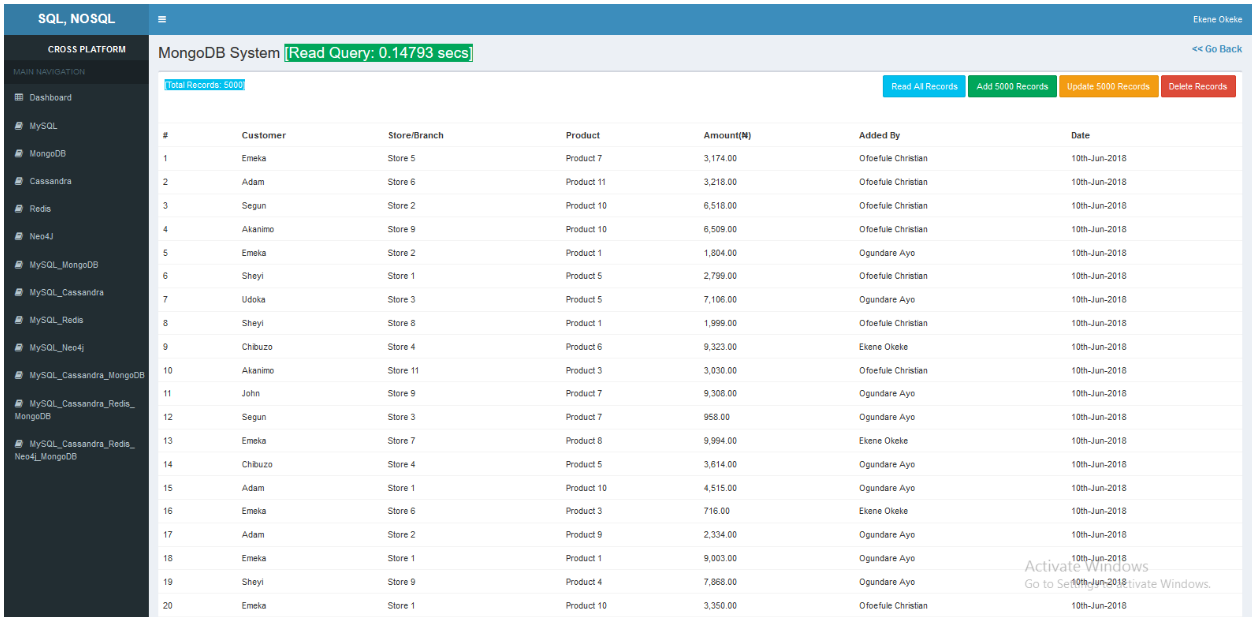
MongoDB System [Add Query: 0.67069 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Emeka	Store 5	Product 7	3,174.00	Ofoefule Christian	10th-Jun-2018
2	Adam	Store 6	Product 11	3,218.00	Ofoefule Christian	10th-Jun-2018
3	Segun	Store 2	Product 10	6,518.00	Ofoefule Christian	10th-Jun-2018
4	Akanimo	Store 9	Product 10	6,509.00	Ofoefule Christian	10th-Jun-2018
5	Emeka	Store 2	Product 1	1,804.00	Ogundare Ayo	10th-Jun-2018
6	Sheyi	Store 1	Product 5	2,799.00	Ofoefule Christian	10th-Jun-2018
7	Udoka	Store 3	Product 5	7,106.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 8	Product 1	1,999.00	Ofoefule Christian	10th-Jun-2018
9	Chibuzo	Store 4	Product 6	9,323.00	Ekene Okeke	10th-Jun-2018
10	Akanimo	Store 11	Product 3	3,030.00	Ofoefule Christian	10th-Jun-2018
11	John	Store 9	Product 7	9,308.00	Ogundare Ayo	10th-Jun-2018
12	Segun	Store 3	Product 7	958.00	Ogundare Ayo	10th-Jun-2018
13	Emeka	Store 7	Product 8	9,994.00	Ekene Okeke	10th-Jun-2018
14	Chibuzo	Store 4	Product 5	3,614.00	Ogundare Ayo	10th-Jun-2018
15	Adam	Store 1	Product 10	4,515.00	Ogundare Ayo	10th-Jun-2018
16	Emeka	Store 6	Product 3	716.00	Ekene Okeke	10th-Jun-2018
17	Adam	Store 2	Product 9	2,334.00	Ogundare Ayo	10th-Jun-2018
18	Emeka	Store 1	Product 1	9,003.00	Ogundare Ayo	10th-Jun-2018
19	Sheyi	Store 9	Product 4	7,868.00	Ogundare Ayo	10th-Jun-2018
20	Emeka	Store 1	Product 10	3,350.00	Ofoefule Christian	10th-Jun-2018

MongoDB Insert Query for 5,000 records

Insert query in MongoDB completed in 0.76069 seconds



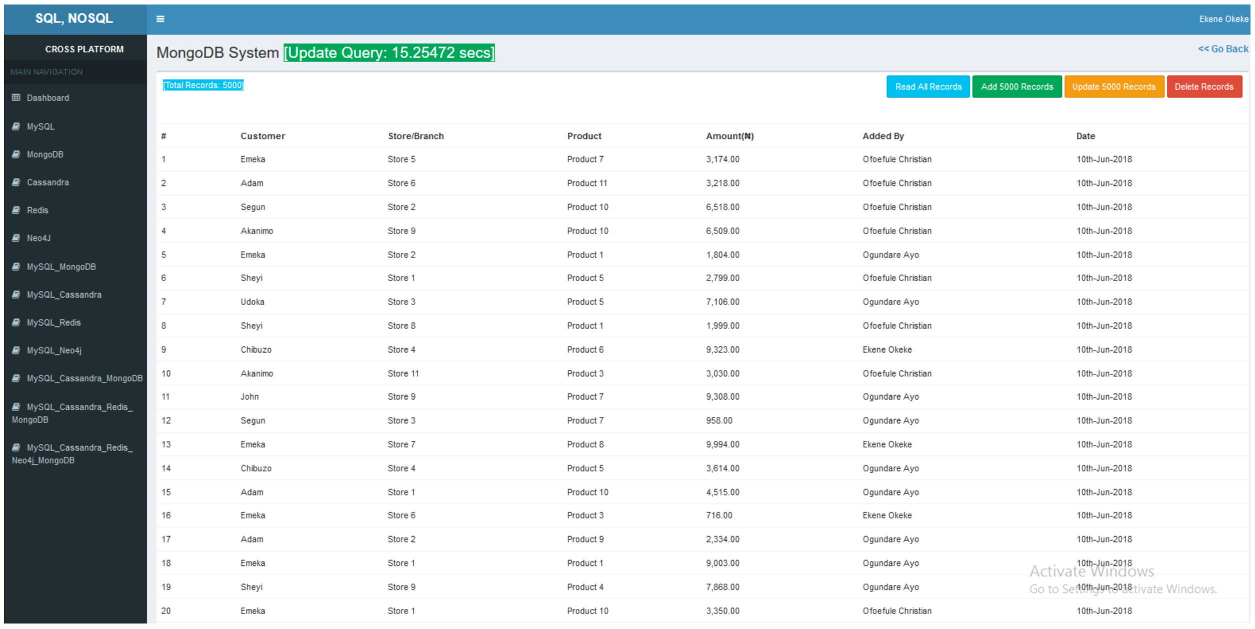
MongoDB System [Read Query: 0.14793 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Emeka	Store 5	Product 7	3,174.00	Ofoefule Christian	10th-Jun-2018
2	Adam	Store 6	Product 11	3,218.00	Ofoefule Christian	10th-Jun-2018
3	Segun	Store 2	Product 10	6,518.00	Ofoefule Christian	10th-Jun-2018
4	Akanimo	Store 9	Product 10	6,509.00	Ofoefule Christian	10th-Jun-2018
5	Emeka	Store 2	Product 1	1,804.00	Ogundare Ayo	10th-Jun-2018
6	Sheyi	Store 1	Product 5	2,799.00	Ofoefule Christian	10th-Jun-2018
7	Udoka	Store 3	Product 5	7,106.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 8	Product 1	1,999.00	Ofoefule Christian	10th-Jun-2018
9	Chibuzo	Store 4	Product 6	9,323.00	Ekene Okeke	10th-Jun-2018
10	Akanimo	Store 11	Product 3	3,030.00	Ofoefule Christian	10th-Jun-2018
11	John	Store 9	Product 7	9,308.00	Ogundare Ayo	10th-Jun-2018
12	Segun	Store 3	Product 7	958.00	Ogundare Ayo	10th-Jun-2018
13	Emeka	Store 7	Product 8	9,994.00	Ekene Okeke	10th-Jun-2018
14	Chibuzo	Store 4	Product 5	3,614.00	Ogundare Ayo	10th-Jun-2018
15	Adam	Store 1	Product 10	4,515.00	Ogundare Ayo	10th-Jun-2018
16	Emeka	Store 6	Product 3	716.00	Ekene Okeke	10th-Jun-2018
17	Adam	Store 2	Product 9	2,334.00	Ogundare Ayo	10th-Jun-2018
18	Emeka	Store 1	Product 1	9,003.00	Ogundare Ayo	10th-Jun-2018
19	Sheyi	Store 9	Product 4	7,868.00	Ogundare Ayo	10th-Jun-2018
20	Emeka	Store 1	Product 10	3,350.00	Ofoefule Christian	10th-Jun-2018

MongoDB Read Query for 5,000 records

Read Query in MongoDB completed in 0.14793 seconds showing that Read operation is fast for MongoDB



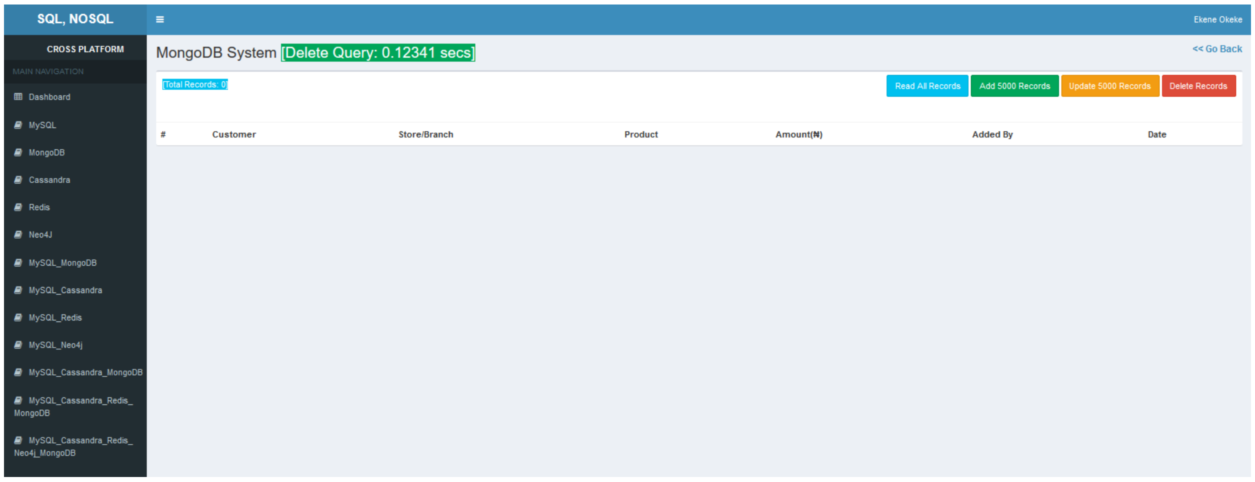
MongoDB System [Update Query: 15.25472 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Emeka	Store 5	Product 7	3,174.00	Ofoefule Christian	10th-Jun-2018
2	Adam	Store 6	Product 11	3,218.00	Ofoefule Christian	10th-Jun-2018
3	Segun	Store 2	Product 10	6,518.00	Ofoefule Christian	10th-Jun-2018
4	Akanimo	Store 9	Product 10	6,509.00	Ofoefule Christian	10th-Jun-2018
5	Emeka	Store 2	Product 1	1,804.00	Ogundare Ayo	10th-Jun-2018
6	Sheyi	Store 1	Product 5	2,799.00	Ofoefule Christian	10th-Jun-2018
7	Udoka	Store 3	Product 5	7,106.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 8	Product 1	1,999.00	Ofoefule Christian	10th-Jun-2018
9	Chibuzo	Store 4	Product 6	9,323.00	Ekene Okeke	10th-Jun-2018
10	Akanimo	Store 11	Product 3	3,030.00	Ofoefule Christian	10th-Jun-2018
11	John	Store 9	Product 7	9,308.00	Ogundare Ayo	10th-Jun-2018
12	Segun	Store 3	Product 7	958.00	Ogundare Ayo	10th-Jun-2018
13	Emeka	Store 7	Product 8	9,994.00	Ekene Okeke	10th-Jun-2018
14	Chibuzo	Store 4	Product 5	3,614.00	Ogundare Ayo	10th-Jun-2018
15	Adam	Store 1	Product 10	4,515.00	Ogundare Ayo	10th-Jun-2018
16	Emeka	Store 6	Product 3	716.00	Ekene Okeke	10th-Jun-2018
17	Adam	Store 2	Product 9	2,334.00	Ogundare Ayo	10th-Jun-2018
18	Emeka	Store 1	Product 1	9,003.00	Ogundare Ayo	10th-Jun-2018
19	Sheyi	Store 9	Product 4	7,868.00	Ogundare Ayo	10th-Jun-2018
20	Emeka	Store 1	Product 10	3,350.00	Ofoefule Christian	10th-Jun-2018

MongoDB Update Query for 5,000 records

Update Query in MongoDb finished in 15.25472 seconds showing that MongoDB is relative fast for Query Update but not as fast as Cassandra.



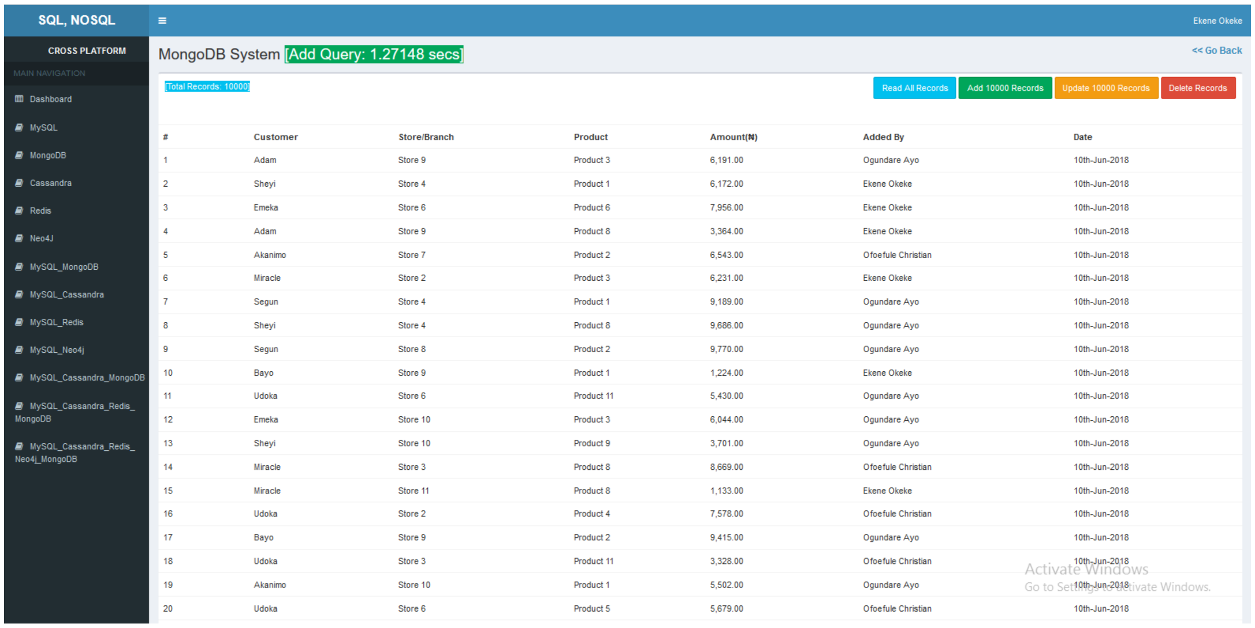
MongoDB System [Delete Query: 0.12341 secs]

[Total Records: 0]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

MongoDB Delete Query for 5,000 records

Delete operation finished at 0.12341 seconds in MongoDB.



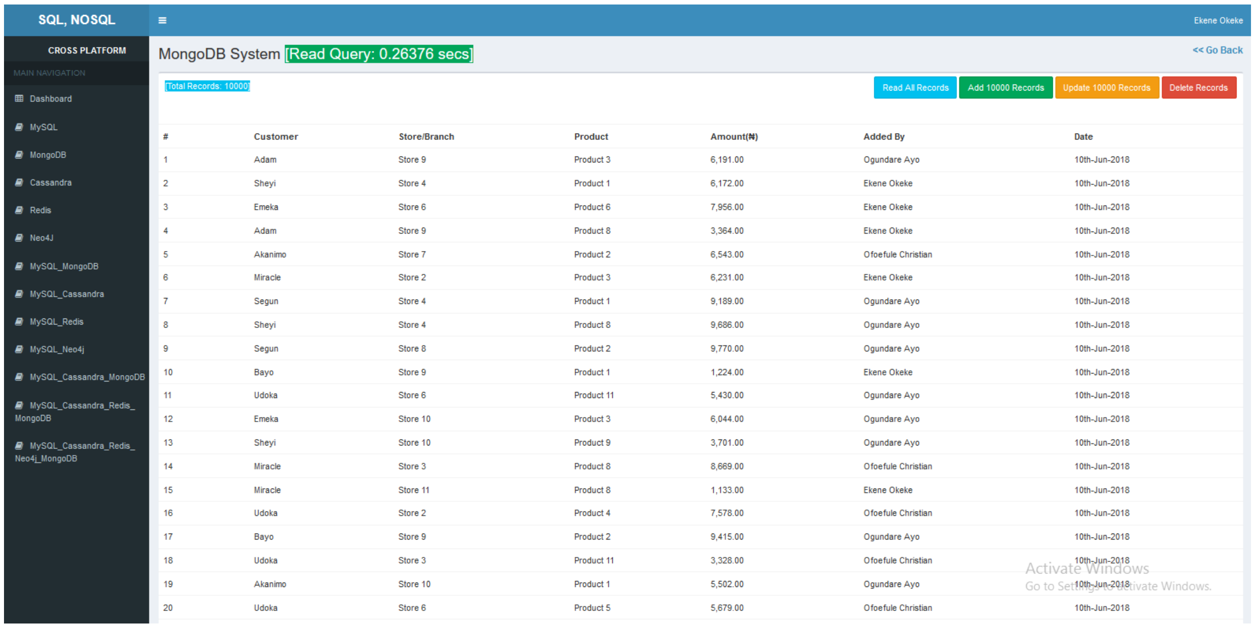
MongoDB System [Add Query: 1.27148 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 9	Product 3	6,191.00	Ogundare Ayo	10th-Jun-2018
2	Sheyi	Store 4	Product 1	6,172.00	Ekene Okeke	10th-Jun-2018
3	Emeka	Store 6	Product 6	7,956.00	Ekene Okeke	10th-Jun-2018
4	Adam	Store 9	Product 8	3,364.00	Ekene Okeke	10th-Jun-2018
5	Akanimo	Store 7	Product 2	6,543.00	Ofoefule Christian	10th-Jun-2018
6	Miracle	Store 2	Product 3	6,231.00	Ekene Okeke	10th-Jun-2018
7	Segun	Store 4	Product 1	9,189.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 4	Product 8	9,686.00	Ogundare Ayo	10th-Jun-2018
9	Segun	Store 8	Product 2	9,770.00	Ogundare Ayo	10th-Jun-2018
10	Bayo	Store 9	Product 1	1,224.00	Ekene Okeke	10th-Jun-2018
11	Udoka	Store 6	Product 11	5,430.00	Ogundare Ayo	10th-Jun-2018
12	Emeka	Store 10	Product 3	6,044.00	Ogundare Ayo	10th-Jun-2018
13	Sheyi	Store 10	Product 9	3,701.00	Ogundare Ayo	10th-Jun-2018
14	Miracle	Store 3	Product 8	8,669.00	Ofoefule Christian	10th-Jun-2018
15	Miracle	Store 11	Product 8	1,133.00	Ekene Okeke	10th-Jun-2018
16	Udoka	Store 2	Product 4	7,578.00	Ofoefule Christian	10th-Jun-2018
17	Bayo	Store 9	Product 2	9,415.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 3	Product 11	3,328.00	Ofoefule Christian	10th-Jun-2018
19	Akanimo	Store 10	Product 1	5,502.00	Ogundare Ayo	10th-Jun-2018
20	Udoka	Store 6	Product 5	5,679.00	Ofoefule Christian	10th-Jun-2018

MongoDB Insert Query for 10,000 records

Insert query finished in 1.27148 seconds for MongoDB which is slower thatn the Redis system which has 0.78274 seconds as its completion time for 10,000 records.



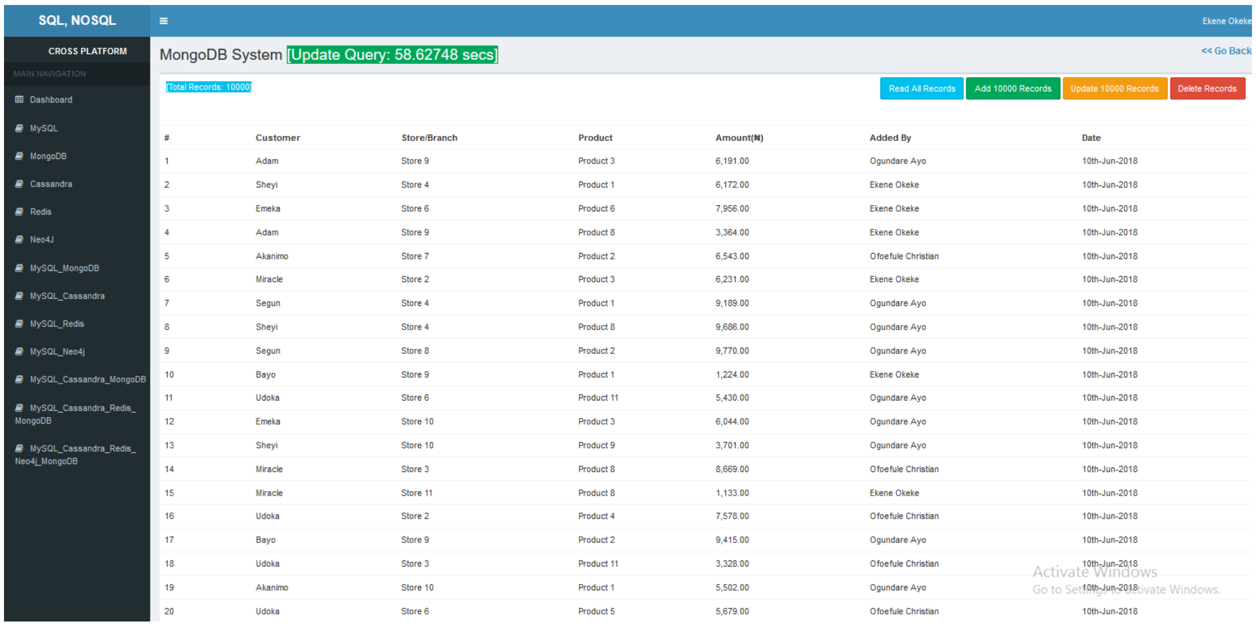
MongoDB System [Read Query: 0.26376 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 9	Product 3	6,191.00	Ogundare Ayo	10th-Jun-2018
2	Sheyi	Store 4	Product 1	6,172.00	Ekene Okeke	10th-Jun-2018
3	Emeka	Store 6	Product 6	7,956.00	Ekene Okeke	10th-Jun-2018
4	Adam	Store 9	Product 8	3,364.00	Ekene Okeke	10th-Jun-2018
5	Akanimo	Store 7	Product 2	6,543.00	Ofoefule Christian	10th-Jun-2018
6	Miracle	Store 2	Product 3	6,231.00	Ekene Okeke	10th-Jun-2018
7	Segun	Store 4	Product 1	9,189.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 4	Product 8	9,686.00	Ogundare Ayo	10th-Jun-2018
9	Segun	Store 8	Product 2	9,770.00	Ogundare Ayo	10th-Jun-2018
10	Bayo	Store 9	Product 1	1,224.00	Ekene Okeke	10th-Jun-2018
11	Udoka	Store 6	Product 11	5,430.00	Ogundare Ayo	10th-Jun-2018
12	Emeka	Store 10	Product 3	6,044.00	Ogundare Ayo	10th-Jun-2018
13	Sheyi	Store 10	Product 9	3,701.00	Ogundare Ayo	10th-Jun-2018
14	Miracle	Store 3	Product 8	8,669.00	Ofoefule Christian	10th-Jun-2018
15	Miracle	Store 11	Product 8	1,133.00	Ekene Okeke	10th-Jun-2018
16	Udoka	Store 2	Product 4	7,578.00	Ofoefule Christian	10th-Jun-2018
17	Bayo	Store 9	Product 2	9,415.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 3	Product 11	3,328.00	Ofoefule Christian	10th-Jun-2018
19	Akanimo	Store 10	Product 1	5,502.00	Ogundare Ayo	10th-Jun-2018
20	Udoka	Store 6	Product 5	5,679.00	Ofoefule Christian	10th-Jun-2018

MongoDB Read Query for 10,000 records

Read query was peformed in 0.26376 seconds in MongoDB. MongoDB is fast for read operations.



MongoDB System [Update Query: 58.62748 secs]

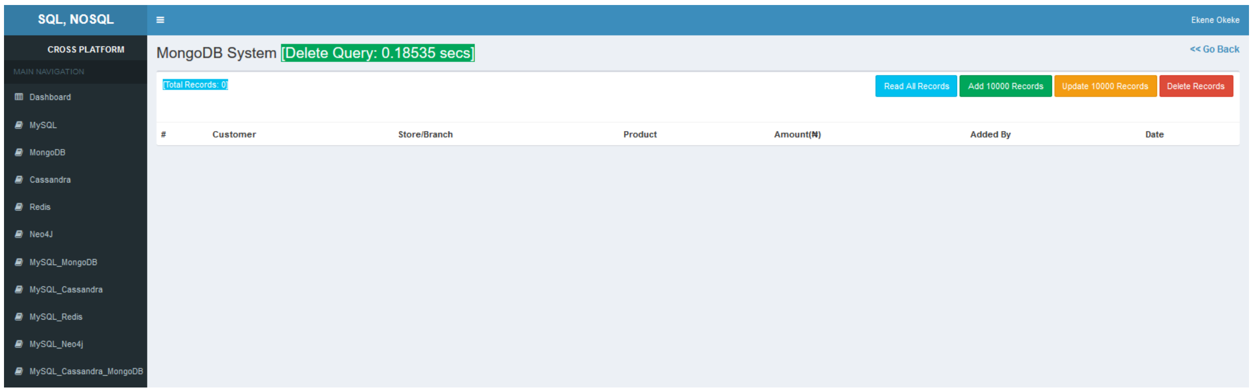
[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 9	Product 3	6,191.00	Ogundare Ayo	10th-Jun-2018
2	Sheyi	Store 4	Product 1	6,172.00	Ekene Okeke	10th-Jun-2018
3	Emeka	Store 6	Product 6	7,956.00	Ekene Okeke	10th-Jun-2018
4	Adam	Store 9	Product 8	3,364.00	Ekene Okeke	10th-Jun-2018
5	Akanimo	Store 7	Product 2	6,543.00	Ofoefule Christian	10th-Jun-2018
6	Miracle	Store 2	Product 3	6,231.00	Ekene Okeke	10th-Jun-2018
7	Segun	Store 4	Product 1	9,189.00	Ogundare Ayo	10th-Jun-2018
8	Sheyi	Store 4	Product 8	9,686.00	Ogundare Ayo	10th-Jun-2018
9	Segun	Store 8	Product 2	9,770.00	Ogundare Ayo	10th-Jun-2018
10	Bayo	Store 9	Product 1	1,224.00	Ekene Okeke	10th-Jun-2018
11	Udoka	Store 6	Product 11	5,430.00	Ogundare Ayo	10th-Jun-2018
12	Emeka	Store 10	Product 3	6,044.00	Ogundare Ayo	10th-Jun-2018
13	Sheyi	Store 10	Product 9	3,701.00	Ogundare Ayo	10th-Jun-2018
14	Miracle	Store 3	Product 8	8,669.00	Ofoefule Christian	10th-Jun-2018
15	Miracle	Store 11	Product 8	1,133.00	Ekene Okeke	10th-Jun-2018
16	Udoka	Store 2	Product 4	7,578.00	Ofoefule Christian	10th-Jun-2018
17	Bayo	Store 9	Product 2	9,415.00	Ogundare Ayo	10th-Jun-2018
18	Udoka	Store 3	Product 11	3,328.00	Ofoefule Christian	10th-Jun-2018
19	Akanimo	Store 10	Product 1	5,502.00	Ogundare Ayo	10th-Jun-2018
20	Udoka	Store 6	Product 5	5,679.00	Ofoefule Christian	10th-Jun-2018

Activate Windows
Go to Settings to activate Windows.

MongoDB Update Query for 10,000 records

Update query for 10,000 records was completed in 58.6275 seconds which is significantly slower that the 12.91777 seconds in the Cassandra system.



MongoDB System [Delete Query: 0.18535 secs]

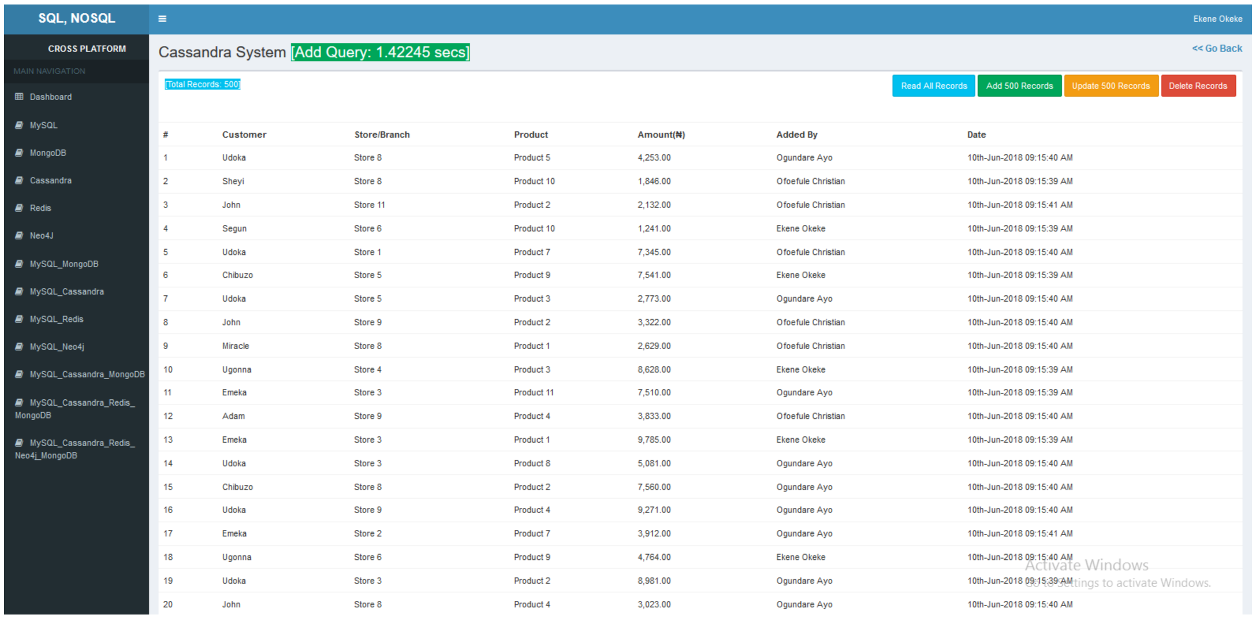
[Total Records: 0]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

MongoDB Delete Query for 10,000 records

Delete Query for MongoDB completed in 0.18535 seconds.

Cassandra System



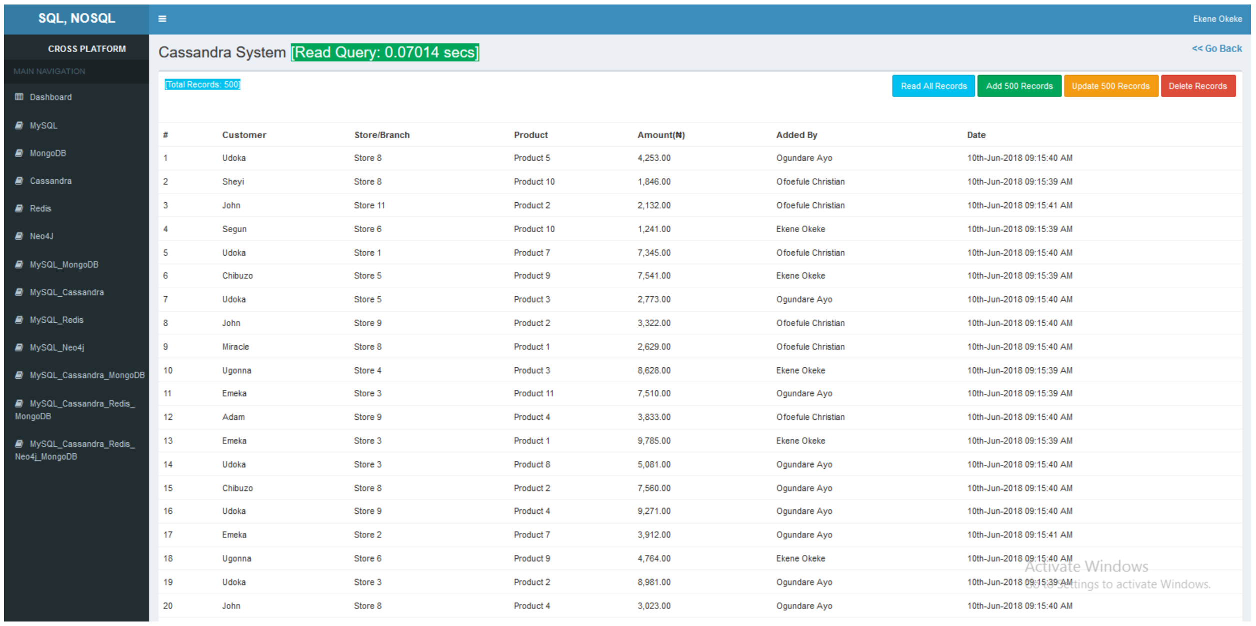
Cassandra System [Add Query: 1.42245 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 8	Product 5	4,253.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
2	Sheyi	Store 8	Product 10	1,846.00	Ofoefule Christian	10th-Jun-2018 09:15:39 AM
3	John	Store 11	Product 2	2,132.00	Ofoefule Christian	10th-Jun-2018 09:15:41 AM
4	Segun	Store 6	Product 10	1,241.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
5	Udoka	Store 1	Product 7	7,345.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
6	Chibuzo	Store 5	Product 9	7,541.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
7	Udoka	Store 5	Product 3	2,773.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
8	John	Store 9	Product 2	3,322.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
9	Miracle	Store 8	Product 1	2,629.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
10	Ugonna	Store 4	Product 3	8,628.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
11	Emeka	Store 3	Product 11	7,510.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
12	Adam	Store 9	Product 4	3,833.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
13	Emeka	Store 3	Product 1	9,785.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
14	Udoka	Store 3	Product 8	5,081.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
15	Chibuzo	Store 8	Product 2	7,560.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
16	Udoka	Store 9	Product 4	9,271.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
17	Emeka	Store 2	Product 7	3,912.00	Ogundare Ayo	10th-Jun-2018 09:15:41 AM
18	Ugonna	Store 6	Product 9	4,764.00	Ekene Okeke	10th-Jun-2018 09:15:40 AM
19	Udoka	Store 3	Product 2	8,981.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
20	John	Store 8	Product 4	3,023.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM

Cassandra Insert Query for 500 records

Add/Insert query completed in 1.42245 seconds for Cassandra. MongoDB performed the same operation for 500 records in0.08013 seconds while Redis completed in 0.13782 seconds (Redis is fastest for insertions).



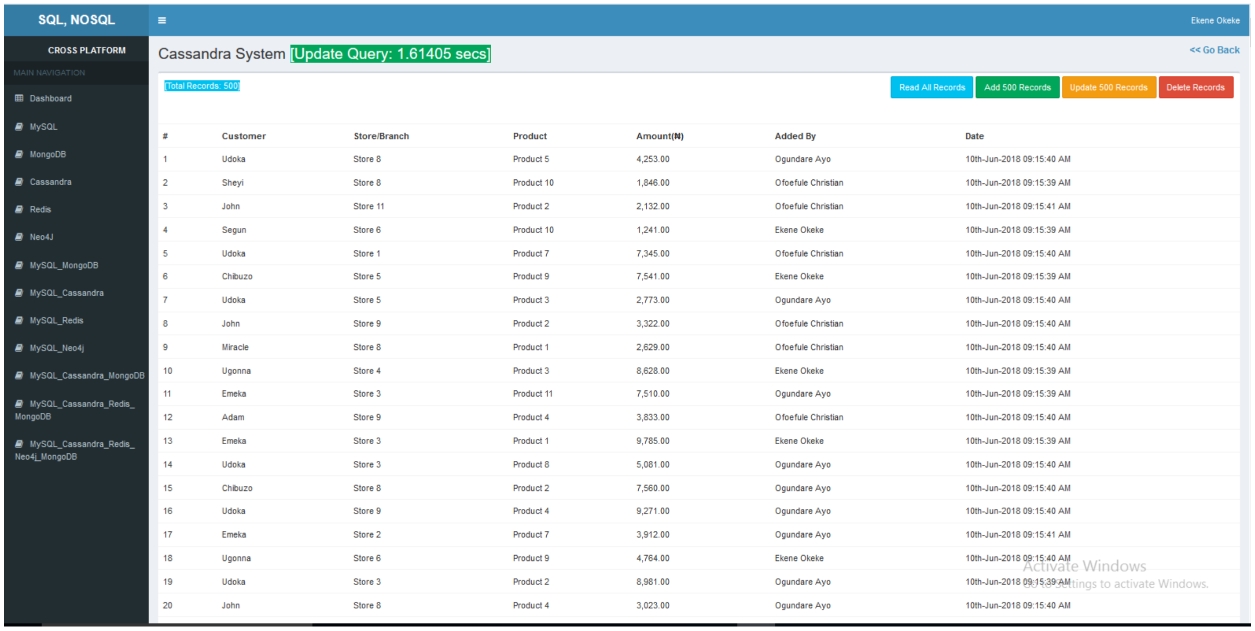
Cassandra System [Read Query: 0.07014 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 8	Product 5	4,253.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
2	Sheyi	Store 8	Product 10	1,846.00	Ofoefule Christian	10th-Jun-2018 09:15:39 AM
3	John	Store 11	Product 2	2,132.00	Ofoefule Christian	10th-Jun-2018 09:15:41 AM
4	Segun	Store 6	Product 10	1,241.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
5	Udoka	Store 1	Product 7	7,345.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
6	Chibuzo	Store 5	Product 9	7,541.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
7	Udoka	Store 5	Product 3	2,773.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
8	John	Store 9	Product 2	3,322.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
9	Miracle	Store 8	Product 1	2,629.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
10	Ugonna	Store 4	Product 3	8,628.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
11	Emeka	Store 3	Product 11	7,510.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
12	Adam	Store 9	Product 4	3,833.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
13	Emeka	Store 3	Product 1	9,785.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
14	Udoka	Store 3	Product 8	5,081.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
15	Chibuzo	Store 8	Product 2	7,560.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
16	Udoka	Store 9	Product 4	9,271.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
17	Emeka	Store 2	Product 7	3,912.00	Ogundare Ayo	10th-Jun-2018 09:15:41 AM
18	Ugonna	Store 6	Product 9	4,764.00	Ekene Okeke	10th-Jun-2018 09:15:40 AM
19	Udoka	Store 3	Product 2	8,981.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
20	John	Store 8	Product 4	3,023.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM

Cassandra Read Query for 500 records

Read query took 0.07014 seconds to complete in Cassandra. MySQL and MongoDB took 0.02148 and 0.01545 seconds to finish.



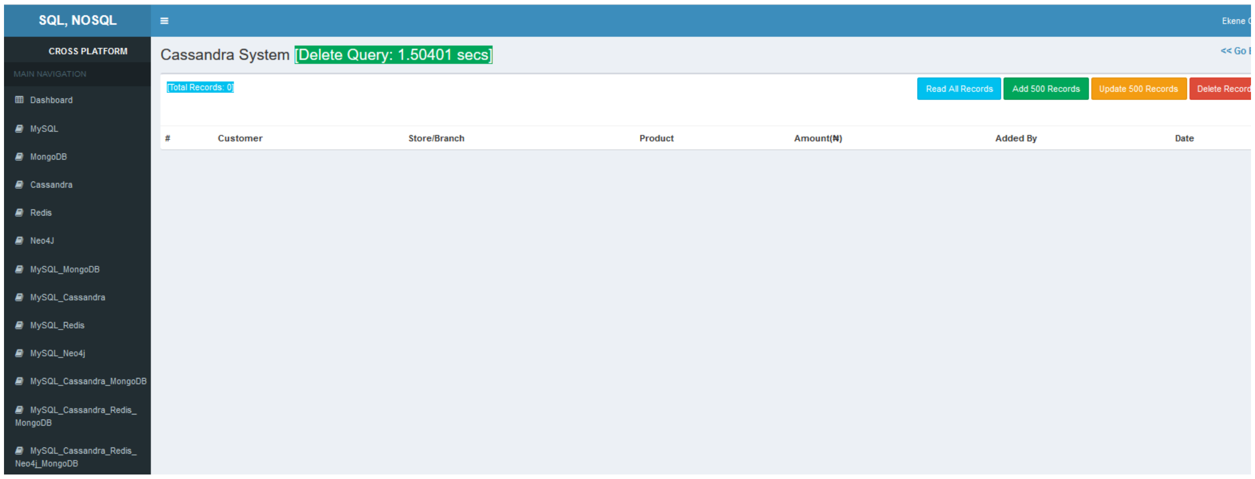
Cassandra System [Update Query: 1.61405 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Udoka	Store 8	Product 5	4,253.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
2	Sheyi	Store 8	Product 10	1,846.00	Ofoefule Christian	10th-Jun-2018 09:15:39 AM
3	John	Store 11	Product 2	2,132.00	Ofoefule Christian	10th-Jun-2018 09:15:41 AM
4	Segun	Store 6	Product 10	1,241.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
5	Udoka	Store 1	Product 7	7,345.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
6	Chibuzo	Store 5	Product 9	7,541.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
7	Udoka	Store 5	Product 3	2,773.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
8	John	Store 9	Product 2	3,322.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
9	Miracle	Store 8	Product 1	2,629.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
10	Ugonna	Store 4	Product 3	8,628.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
11	Emeka	Store 3	Product 11	7,510.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
12	Adam	Store 9	Product 4	3,833.00	Ofoefule Christian	10th-Jun-2018 09:15:40 AM
13	Emeka	Store 3	Product 1	9,785.00	Ekene Okeke	10th-Jun-2018 09:15:39 AM
14	Udoka	Store 3	Product 8	5,081.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
15	Chibuzo	Store 8	Product 2	7,560.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
16	Udoka	Store 9	Product 4	9,271.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM
17	Emeka	Store 2	Product 7	3,912.00	Ogundare Ayo	10th-Jun-2018 09:15:41 AM
18	Ugonna	Store 6	Product 9	4,764.00	Ekene Okeke	10th-Jun-2018 09:15:40 AM
19	Udoka	Store 3	Product 2	8,981.00	Ogundare Ayo	10th-Jun-2018 09:15:39 AM
20	John	Store 8	Product 4	3,023.00	Ogundare Ayo	10th-Jun-2018 09:15:40 AM

Cassandra Update Query for 500 records

Update query was completed in 1.61405 seconds for Cassandra. Cassandra is write-intensive and performs updates fast for large records.



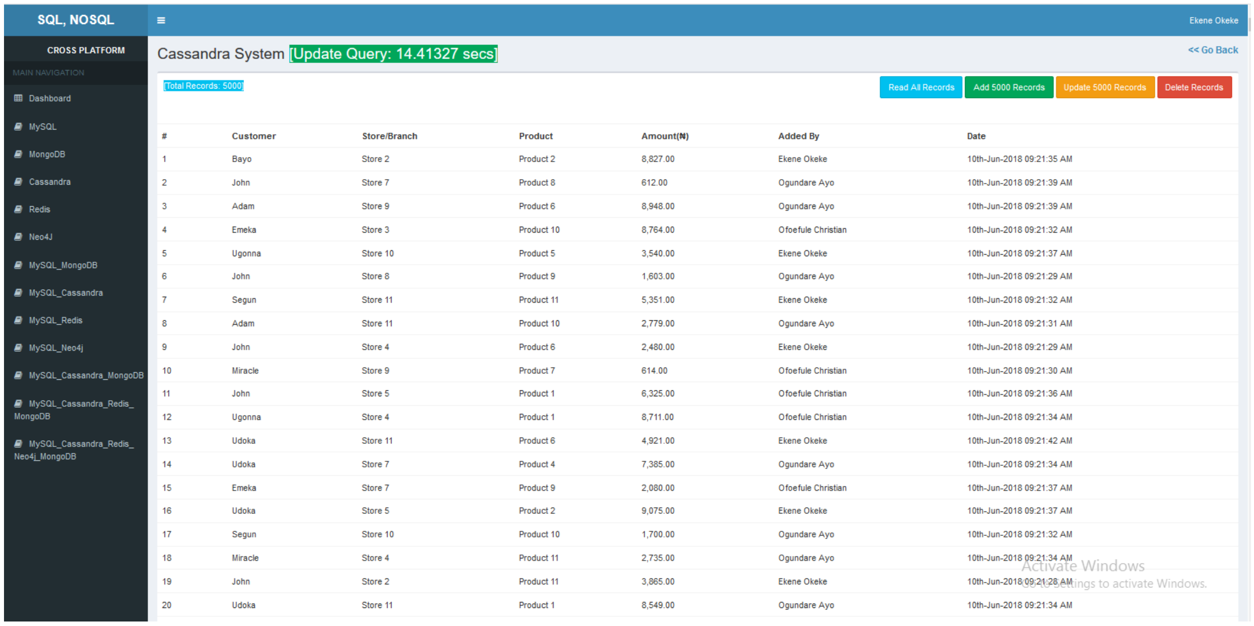
Cassandra System [Delete Query: 1.50401 secs]

[Total Records: 0]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Cassandra Delete Query for 500 records

Delete operation was completed in 1.50401 seconds in Cassandra.



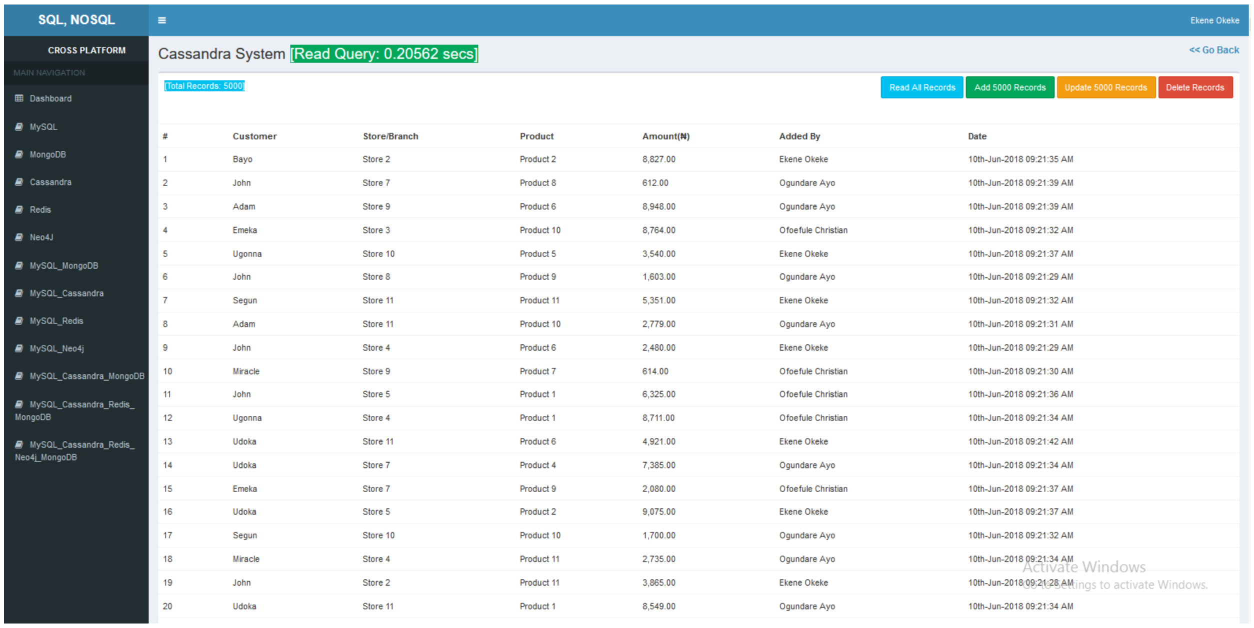
Cassandra System [Update Query: 14.41327 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Bayo	Store 2	Product 2	8,827.00	Ekene Okeke	10th-Jun-2018 09:21:35 AM
2	John	Store 7	Product 8	612.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
3	Adam	Store 9	Product 6	8,948.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
4	Emeka	Store 3	Product 10	8,764.00	Ofoefule Christian	10th-Jun-2018 09:21:32 AM
5	Ugonna	Store 10	Product 5	3,540.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
6	John	Store 8	Product 9	1,603.00	Ogundare Ayo	10th-Jun-2018 09:21:29 AM
7	Segun	Store 11	Product 11	5,351.00	Ekene Okeke	10th-Jun-2018 09:21:32 AM
8	Adam	Store 11	Product 10	2,779.00	Ogundare Ayo	10th-Jun-2018 09:21:31 AM
9	John	Store 4	Product 6	2,480.00	Ekene Okeke	10th-Jun-2018 09:21:29 AM
10	Miracle	Store 9	Product 7	614.00	Ofoefule Christian	10th-Jun-2018 09:21:30 AM
11	John	Store 5	Product 1	6,325.00	Ofoefule Christian	10th-Jun-2018 09:21:36 AM
12	Ugonna	Store 4	Product 1	8,711.00	Ofoefule Christian	10th-Jun-2018 09:21:34 AM
13	Udoka	Store 11	Product 6	4,921.00	Ekene Okeke	10th-Jun-2018 09:21:42 AM
14	Udoka	Store 7	Product 4	7,385.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
15	Emeka	Store 7	Product 9	2,080.00	Ofoefule Christian	10th-Jun-2018 09:21:37 AM
16	Udoka	Store 5	Product 2	9,075.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
17	Segun	Store 10	Product 10	1,700.00	Ogundare Ayo	10th-Jun-2018 09:21:32 AM
18	Miracle	Store 4	Product 11	2,735.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
19	John	Store 2	Product 11	3,865.00	Ekene Okeke	10th-Jun-2018 09:21:28 AM
20	Udoka	Store 11	Product 1	8,549.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM

Cassandra Insert Query for 5,000 records

Cassandra completed insertion of 5,000 records in 14.525 seconds while Redis did the same in 0.3924 seconds and MongoDB 0.6707 seconds; MySQL perfomed the same operation in 212.86839 seconds.



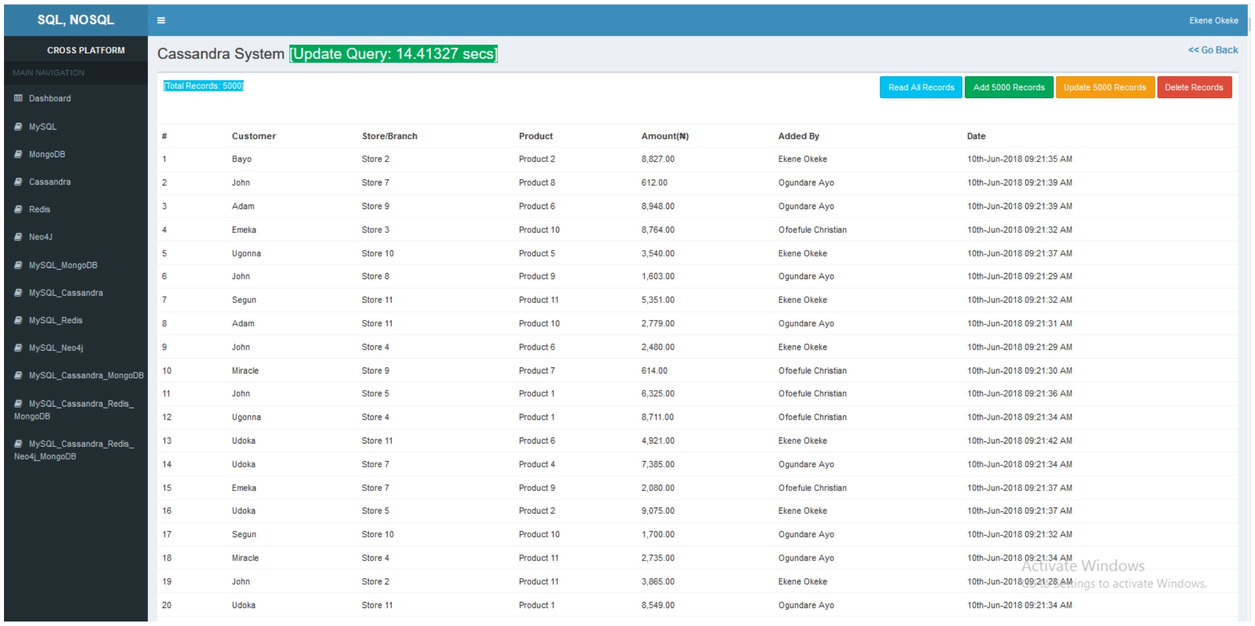
Cassandra System [Read Query: 0.20562 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Bayo	Store 2	Product 2	8,827.00	Ekene Okeke	10th-Jun-2018 09:21:35 AM
2	John	Store 7	Product 8	612.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
3	Adam	Store 9	Product 6	8,948.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
4	Emeka	Store 3	Product 10	8,764.00	Ofoefule Christian	10th-Jun-2018 09:21:32 AM
5	Ugonna	Store 10	Product 5	3,540.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
6	John	Store 8	Product 9	1,603.00	Ogundare Ayo	10th-Jun-2018 09:21:29 AM
7	Segun	Store 11	Product 11	5,351.00	Ekene Okeke	10th-Jun-2018 09:21:32 AM
8	Adam	Store 11	Product 10	2,779.00	Ogundare Ayo	10th-Jun-2018 09:21:31 AM
9	John	Store 4	Product 6	2,480.00	Ekene Okeke	10th-Jun-2018 09:21:29 AM
10	Miracle	Store 9	Product 7	614.00	Ofoefule Christian	10th-Jun-2018 09:21:30 AM
11	John	Store 5	Product 1	6,325.00	Ofoefule Christian	10th-Jun-2018 09:21:36 AM
12	Ugonna	Store 4	Product 1	8,711.00	Ofoefule Christian	10th-Jun-2018 09:21:34 AM
13	Udoka	Store 11	Product 6	4,921.00	Ekene Okeke	10th-Jun-2018 09:21:42 AM
14	Udoka	Store 7	Product 4	7,385.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
15	Emeka	Store 7	Product 9	2,080.00	Ofoefule Christian	10th-Jun-2018 09:21:37 AM
16	Udoka	Store 5	Product 2	9,075.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
17	Segun	Store 10	Product 10	1,700.00	Ogundare Ayo	10th-Jun-2018 09:21:32 AM
18	Miracle	Store 4	Product 11	2,735.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
19	John	Store 2	Product 11	3,865.00	Ekene Okeke	10th-Jun-2018 09:21:28 AM
20	Udoka	Store 11	Product 1	8,549.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM

Cassandra Read Query for 5,000 records

Cassandra completed the Read operation in 0.20562 seconds which is not as fast as MySQL which has completion time of 0.18324 seonds and MongoDB 0.14793 seconds.



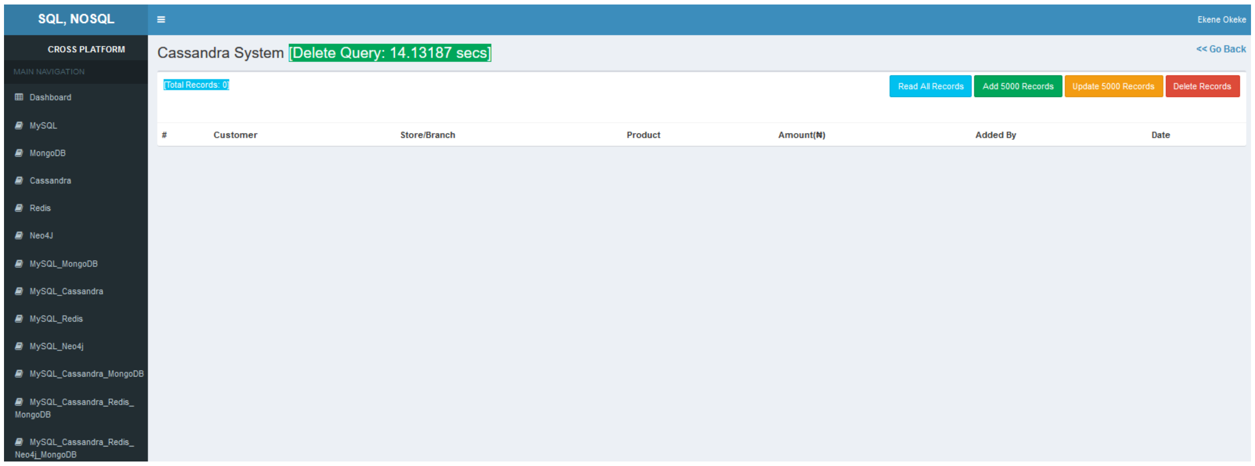
Cassandra System [Update Query: 14.41327 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Bayo	Store 2	Product 2	8,827.00	Ekene Okeke	10th-Jun-2018 09:21:35 AM
2	John	Store 7	Product 8	612.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
3	Adam	Store 9	Product 6	8,948.00	Ogundare Ayo	10th-Jun-2018 09:21:39 AM
4	Emeka	Store 3	Product 10	8,764.00	Ofoefule Christian	10th-Jun-2018 09:21:32 AM
5	Ugonna	Store 10	Product 5	3,540.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
6	John	Store 8	Product 9	1,603.00	Ogundare Ayo	10th-Jun-2018 09:21:29 AM
7	Segun	Store 11	Product 11	5,351.00	Ekene Okeke	10th-Jun-2018 09:21:32 AM
8	Adam	Store 11	Product 10	2,779.00	Ogundare Ayo	10th-Jun-2018 09:21:31 AM
9	John	Store 4	Product 6	2,480.00	Ekene Okeke	10th-Jun-2018 09:21:29 AM
10	Miracle	Store 9	Product 7	614.00	Ofoefule Christian	10th-Jun-2018 09:21:30 AM
11	John	Store 5	Product 1	6,325.00	Ofoefule Christian	10th-Jun-2018 09:21:36 AM
12	Ugonna	Store 4	Product 1	8,711.00	Ofoefule Christian	10th-Jun-2018 09:21:34 AM
13	Udoka	Store 11	Product 6	4,921.00	Ekene Okeke	10th-Jun-2018 09:21:42 AM
14	Udoka	Store 7	Product 4	7,385.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
15	Emeka	Store 7	Product 9	2,080.00	Ofoefule Christian	10th-Jun-2018 09:21:37 AM
16	Udoka	Store 5	Product 2	9,075.00	Ekene Okeke	10th-Jun-2018 09:21:37 AM
17	Segun	Store 10	Product 10	1,700.00	Ogundare Ayo	10th-Jun-2018 09:21:32 AM
18	Miracle	Store 4	Product 11	2,735.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM
19	John	Store 2	Product 11	3,865.00	Ekene Okeke	10th-Jun-2018 09:21:28 AM
20	Udoka	Store 11	Product 1	8,549.00	Ogundare Ayo	10th-Jun-2018 09:21:34 AM

Cassandra Update Query for 5,000 records

Update Query was completed in 14.41327 seconds. MongoDB performed the same operation in 15.255 seconds. Cassandra is best fit for update operation.



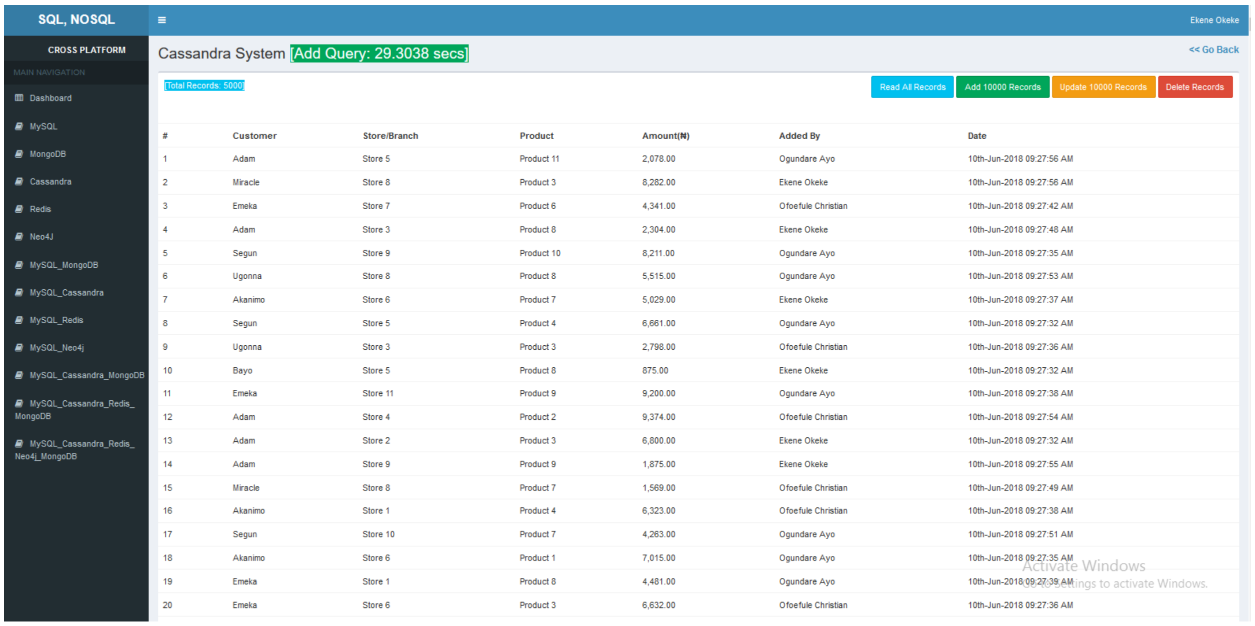
Cassandra System [Delete Query: 14.13187 secs]

[Total Records: 0]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Cassandra Delete Query for 5,000 records

Delete query completed in 14.1387 seconds for Cassandra system. Cassandra is slow for delete purpose.



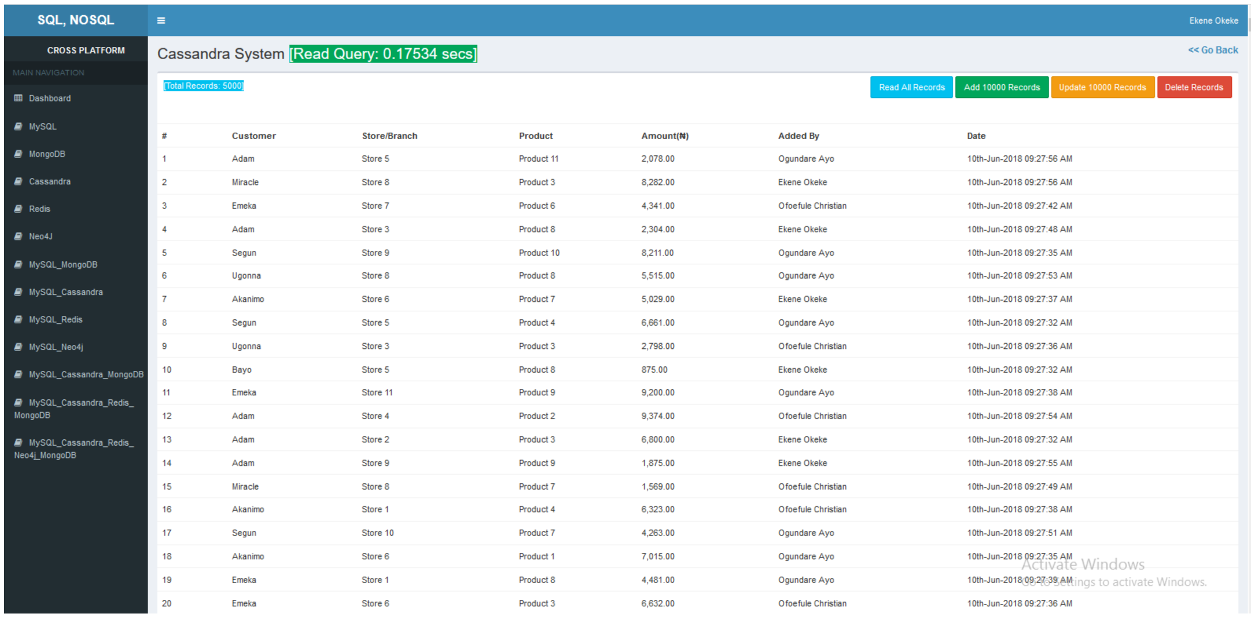
Cassandra System [Add Query: 29.3038 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 5	Product 11	2,078.00	Ogundare Ayo	10th-Jun-2018 09:27:56 AM
2	Miracle	Store 8	Product 3	8,282.00	Ekene Okeke	10th-Jun-2018 09:27:56 AM
3	Emeka	Store 7	Product 6	4,341.00	Ofoefule Christian	10th-Jun-2018 09:27:42 AM
4	Adam	Store 3	Product 8	2,304.00	Ekene Okeke	10th-Jun-2018 09:27:48 AM
5	Segun	Store 9	Product 10	8,211.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
6	Ugonna	Store 8	Product 8	5,515.00	Ogundare Ayo	10th-Jun-2018 09:27:53 AM
7	Akanimo	Store 6	Product 7	5,029.00	Ekene Okeke	10th-Jun-2018 09:27:37 AM
8	Segun	Store 5	Product 4	6,661.00	Ogundare Ayo	10th-Jun-2018 09:27:32 AM
9	Ugonna	Store 3	Product 3	2,798.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM
10	Bayo	Store 5	Product 8	875.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
11	Emeka	Store 11	Product 9	9,200.00	Ogundare Ayo	10th-Jun-2018 09:27:38 AM
12	Adam	Store 4	Product 2	9,374.00	Ofoefule Christian	10th-Jun-2018 09:27:54 AM
13	Adam	Store 2	Product 3	6,800.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
14	Adam	Store 9	Product 9	1,875.00	Ekene Okeke	10th-Jun-2018 09:27:55 AM
15	Miracle	Store 8	Product 7	1,569.00	Ofoefule Christian	10th-Jun-2018 09:27:49 AM
16	Akanimo	Store 1	Product 4	6,323.00	Ofoefule Christian	10th-Jun-2018 09:27:38 AM
17	Segun	Store 10	Product 7	4,263.00	Ogundare Ayo	10th-Jun-2018 09:27:51 AM
18	Akanimo	Store 6	Product 1	7,015.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
19	Emeka	Store 1	Product 8	4,481.00	Ogundare Ayo	10th-Jun-2018 09:27:39 AM
20	Emeka	Store 6	Product 3	6,632.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM

Cassandra Insert Query for 10,000 records

Add query took 29.3038 seconds to complete in Cassandra system. Cassandra is not the best option for insertion purpose in the hybrid system.



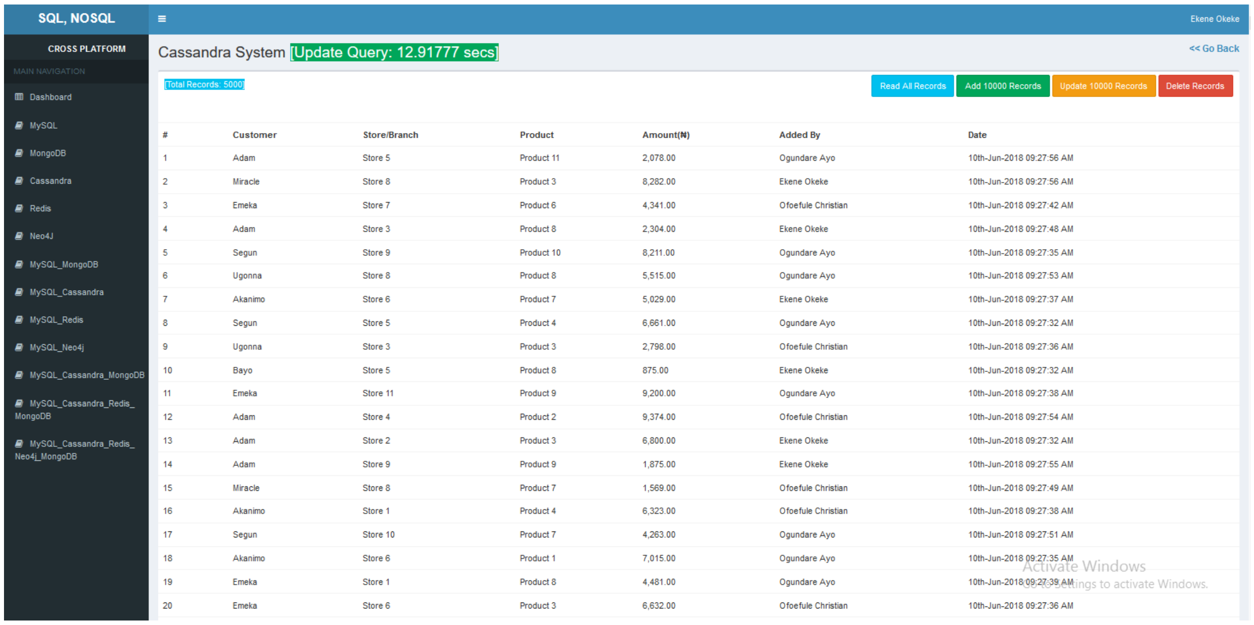
Cassandra System [Read Query: 0.17534 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 5	Product 11	2,078.00	Ogundare Ayo	10th-Jun-2018 09:27:56 AM
2	Miracle	Store 8	Product 3	8,282.00	Ekene Okeke	10th-Jun-2018 09:27:56 AM
3	Emeka	Store 7	Product 6	4,341.00	Ofoefule Christian	10th-Jun-2018 09:27:42 AM
4	Adam	Store 3	Product 8	2,304.00	Ekene Okeke	10th-Jun-2018 09:27:48 AM
5	Segun	Store 9	Product 10	8,211.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
6	Ugonna	Store 8	Product 8	5,515.00	Ogundare Ayo	10th-Jun-2018 09:27:53 AM
7	Akanimo	Store 6	Product 7	5,029.00	Ekene Okeke	10th-Jun-2018 09:27:37 AM
8	Segun	Store 5	Product 4	6,661.00	Ogundare Ayo	10th-Jun-2018 09:27:32 AM
9	Ugonna	Store 3	Product 3	2,798.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM
10	Bayo	Store 5	Product 8	875.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
11	Emeka	Store 11	Product 9	9,200.00	Ogundare Ayo	10th-Jun-2018 09:27:38 AM
12	Adam	Store 4	Product 2	9,374.00	Ofoefule Christian	10th-Jun-2018 09:27:54 AM
13	Adam	Store 2	Product 3	6,800.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
14	Adam	Store 9	Product 9	1,875.00	Ekene Okeke	10th-Jun-2018 09:27:55 AM
15	Miracle	Store 8	Product 7	1,569.00	Ofoefule Christian	10th-Jun-2018 09:27:49 AM
16	Akanimo	Store 1	Product 4	6,323.00	Ofoefule Christian	10th-Jun-2018 09:27:38 AM
17	Segun	Store 10	Product 7	4,263.00	Ogundare Ayo	10th-Jun-2018 09:27:51 AM
18	Akanimo	Store 6	Product 1	7,015.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
19	Emeka	Store 1	Product 8	4,481.00	Ogundare Ayo	10th-Jun-2018 09:27:39 AM
20	Emeka	Store 6	Product 3	6,632.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM

Cassandra Read Query for 10,000 records

Read query was completed in 0.17534 seconds for Cassandra system. MongoDB and MySQL performed better for the same operation using the same amount of records.



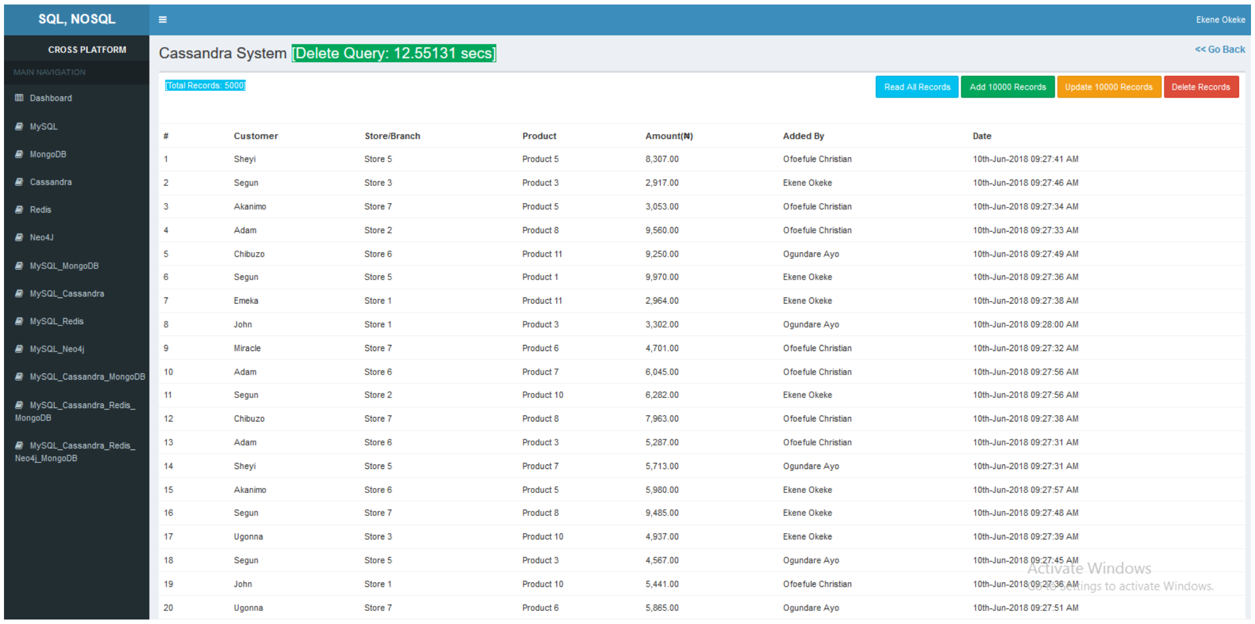
Cassandra System [Update Query: 12.91777 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 5	Product 11	2,078.00	Ogundare Ayo	10th-Jun-2018 09:27:56 AM
2	Miracle	Store 8	Product 3	8,282.00	Ekene Okeke	10th-Jun-2018 09:27:56 AM
3	Emeka	Store 7	Product 6	4,341.00	Ofoefule Christian	10th-Jun-2018 09:27:42 AM
4	Adam	Store 3	Product 8	2,304.00	Ekene Okeke	10th-Jun-2018 09:27:48 AM
5	Segun	Store 9	Product 10	8,211.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
6	Ugonna	Store 8	Product 8	5,515.00	Ogundare Ayo	10th-Jun-2018 09:27:53 AM
7	Akanimo	Store 6	Product 7	5,029.00	Ekene Okeke	10th-Jun-2018 09:27:37 AM
8	Segun	Store 5	Product 4	6,661.00	Ogundare Ayo	10th-Jun-2018 09:27:32 AM
9	Ugonna	Store 3	Product 3	2,798.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM
10	Bayo	Store 5	Product 8	875.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
11	Emeka	Store 11	Product 9	9,200.00	Ogundare Ayo	10th-Jun-2018 09:27:38 AM
12	Adam	Store 4	Product 2	9,374.00	Ofoefule Christian	10th-Jun-2018 09:27:54 AM
13	Adam	Store 2	Product 3	6,800.00	Ekene Okeke	10th-Jun-2018 09:27:32 AM
14	Adam	Store 9	Product 9	1,875.00	Ekene Okeke	10th-Jun-2018 09:27:55 AM
15	Miracle	Store 8	Product 7	1,569.00	Ofoefule Christian	10th-Jun-2018 09:27:49 AM
16	Akanimo	Store 1	Product 4	6,323.00	Ofoefule Christian	10th-Jun-2018 09:27:38 AM
17	Segun	Store 10	Product 7	4,263.00	Ogundare Ayo	10th-Jun-2018 09:27:51 AM
18	Akanimo	Store 6	Product 1	7,015.00	Ogundare Ayo	10th-Jun-2018 09:27:35 AM
19	Emeka	Store 1	Product 8	4,481.00	Ogundare Ayo	10th-Jun-2018 09:27:39 AM
20	Emeka	Store 6	Product 3	6,632.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM

Cassandra Update Query for 10,000 records

Update query completed in 12.91777 sseconds in Cassandra. Cassandra is very fast for large volume of update operation.



Cassandra System [Delete Query: 12.55131 secs]

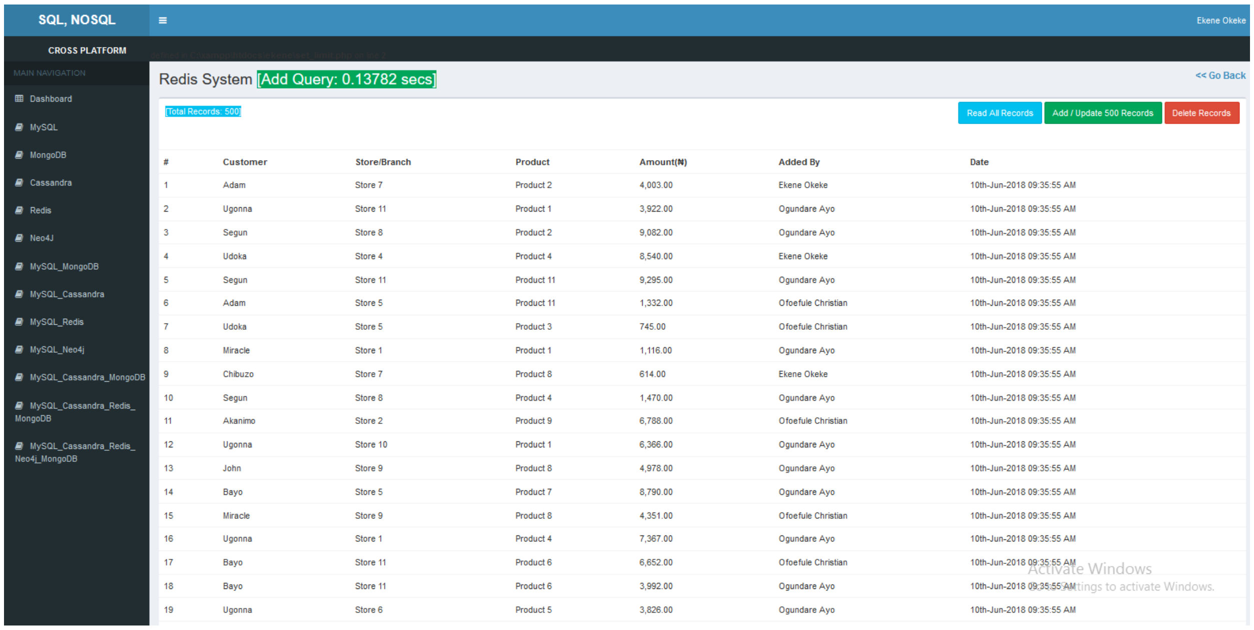
[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 5	Product 5	8,307.00	Ofoefule Christian	10th-Jun-2018 09:27:41 AM
2	Segun	Store 3	Product 3	2,917.00	Ekene Okeke	10th-Jun-2018 09:27:46 AM
3	Akanimo	Store 7	Product 5	3,053.00	Ofoefule Christian	10th-Jun-2018 09:27:34 AM
4	Adam	Store 2	Product 8	9,560.00	Ofoefule Christian	10th-Jun-2018 09:27:33 AM
5	Chibuzo	Store 6	Product 11	9,250.00	Ogundare Ayo	10th-Jun-2018 09:27:49 AM
6	Segun	Store 5	Product 1	9,970.00	Ekene Okeke	10th-Jun-2018 09:27:36 AM
7	Emeka	Store 1	Product 11	2,964.00	Ekene Okeke	10th-Jun-2018 09:27:38 AM
8	John	Store 1	Product 3	3,302.00	Ogundare Ayo	10th-Jun-2018 09:28:00 AM
9	Miracle	Store 7	Product 6	4,701.00	Ofoefule Christian	10th-Jun-2018 09:27:32 AM
10	Adam	Store 6	Product 7	6,045.00	Ofoefule Christian	10th-Jun-2018 09:27:56 AM
11	Segun	Store 2	Product 10	6,282.00	Ekene Okeke	10th-Jun-2018 09:27:56 AM
12	Chibuzo	Store 7	Product 8	7,963.00	Ofoefule Christian	10th-Jun-2018 09:27:38 AM
13	Adam	Store 6	Product 3	5,287.00	Ofoefule Christian	10th-Jun-2018 09:27:31 AM
14	Sheyi	Store 5	Product 7	5,713.00	Ogundare Ayo	10th-Jun-2018 09:27:31 AM
15	Akanimo	Store 6	Product 5	5,980.00	Ekene Okeke	10th-Jun-2018 09:27:57 AM
16	Segun	Store 7	Product 8	9,485.00	Ekene Okeke	10th-Jun-2018 09:27:48 AM
17	Ugonna	Store 3	Product 10	4,937.00	Ekene Okeke	10th-Jun-2018 09:27:39 AM
18	Segun	Store 5	Product 3	4,567.00	Ogundare Ayo	10th-Jun-2018 09:27:45 AM
19	John	Store 1	Product 10	5,441.00	Ofoefule Christian	10th-Jun-2018 09:27:36 AM
20	Ugonna	Store 7	Product 6	5,865.00	Ogundare Ayo	10th-Jun-2018 09:27:51 AM

Cassandra Delete Query for 10,000 records

Delete query completed in 12.55131 seconds.

Redis System



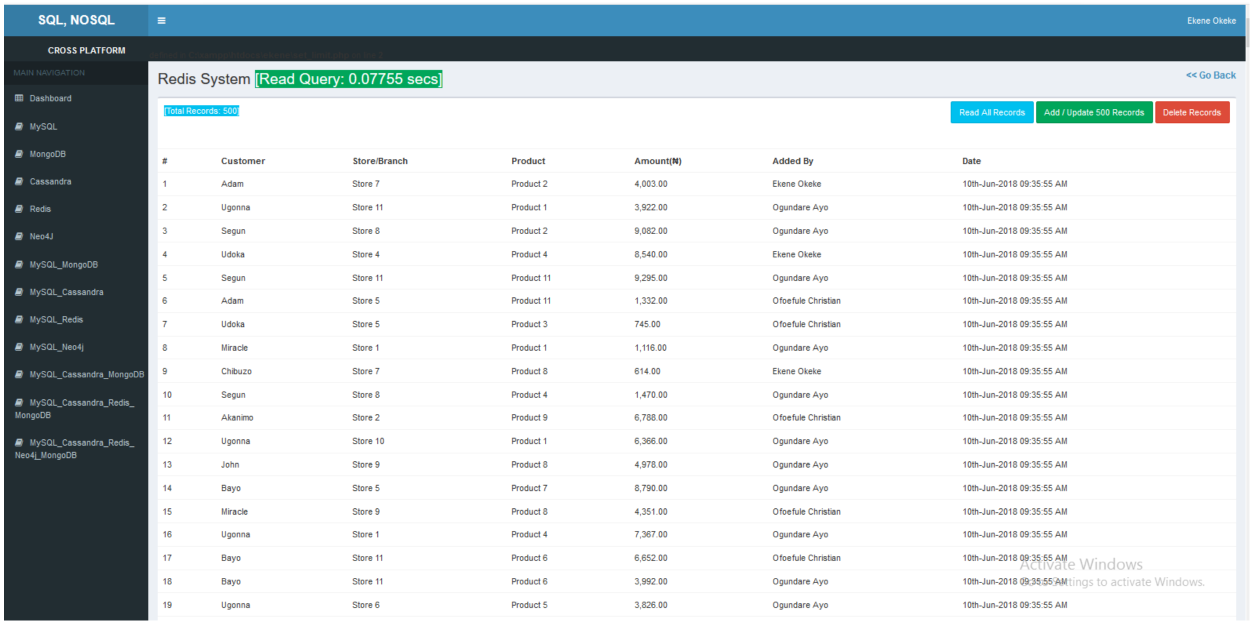
Redis System [Add Query: 0.13782 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 7	Product 2	4,003.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
2	Ugonna	Store 11	Product 1	3,922.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
3	Segun	Store 8	Product 2	9,082.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
4	Udoka	Store 4	Product 4	8,540.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
5	Segun	Store 11	Product 11	9,295.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
6	Adam	Store 5	Product 11	1,332.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
7	Udoka	Store 5	Product 3	745.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
8	Miracle	Store 1	Product 1	1,116.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
9	Chibuzo	Store 7	Product 8	614.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
10	Segun	Store 8	Product 4	1,470.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
11	Akanimo	Store 2	Product 9	6,788.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
12	Ugonna	Store 10	Product 1	6,366.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
13	John	Store 9	Product 8	4,978.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
14	Bayo	Store 5	Product 7	8,790.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
15	Miracle	Store 9	Product 8	4,351.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
16	Ugonna	Store 1	Product 4	7,367.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
17	Bayo	Store 11	Product 6	6,652.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
18	Bayo	Store 11	Product 6	3,992.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
19	Ugonna	Store 6	Product 5	3,826.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM

Redis Insert Query for 500 records

Insert query had completion time of 0.13782 seconds which is faster than all the other systems for the same number of records. MySQL performed insertion of 500 records in 21.33679 seconds showing that MySQL is very slow for insert operations.



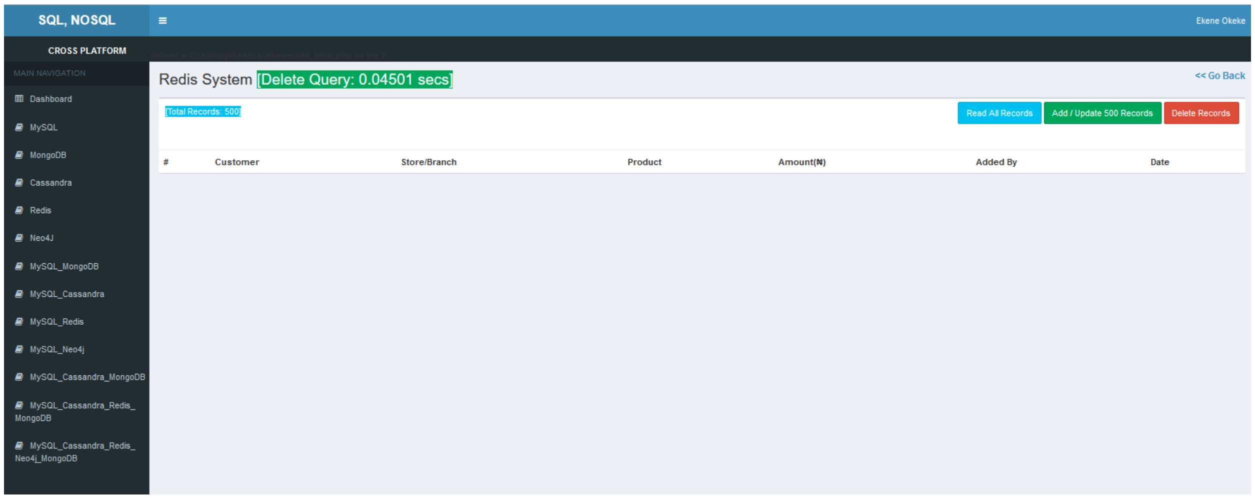
Redis System [Read Query: 0.07755 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Adam	Store 7	Product 2	4,003.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
2	Ugonna	Store 11	Product 1	3,922.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
3	Segun	Store 8	Product 2	9,082.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
4	Udoka	Store 4	Product 4	8,540.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
5	Segun	Store 11	Product 11	9,295.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
6	Adam	Store 5	Product 11	1,332.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
7	Udoka	Store 5	Product 3	745.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
8	Miracle	Store 1	Product 1	1,116.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
9	Chibuzo	Store 7	Product 8	614.00	Ekene Okeke	10th-Jun-2018 09:35:55 AM
10	Segun	Store 8	Product 4	1,470.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
11	Akanimo	Store 2	Product 9	6,788.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
12	Ugonna	Store 10	Product 1	6,366.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
13	John	Store 9	Product 8	4,978.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
14	Bayo	Store 5	Product 7	8,790.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
15	Miracle	Store 9	Product 8	4,351.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
16	Ugonna	Store 1	Product 4	7,367.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
17	Bayo	Store 11	Product 6	6,652.00	Ofoefule Christian	10th-Jun-2018 09:35:55 AM
18	Bayo	Store 11	Product 6	3,992.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM
19	Ugonna	Store 6	Product 5	3,826.00	Ogundare Ayo	10th-Jun-2018 09:35:55 AM

Redis Read Query for 500 records

Read query shows 0.077155 seconds as the completion time. This falls behind MongoDBand MySQL for the same operation, using the same number of records.



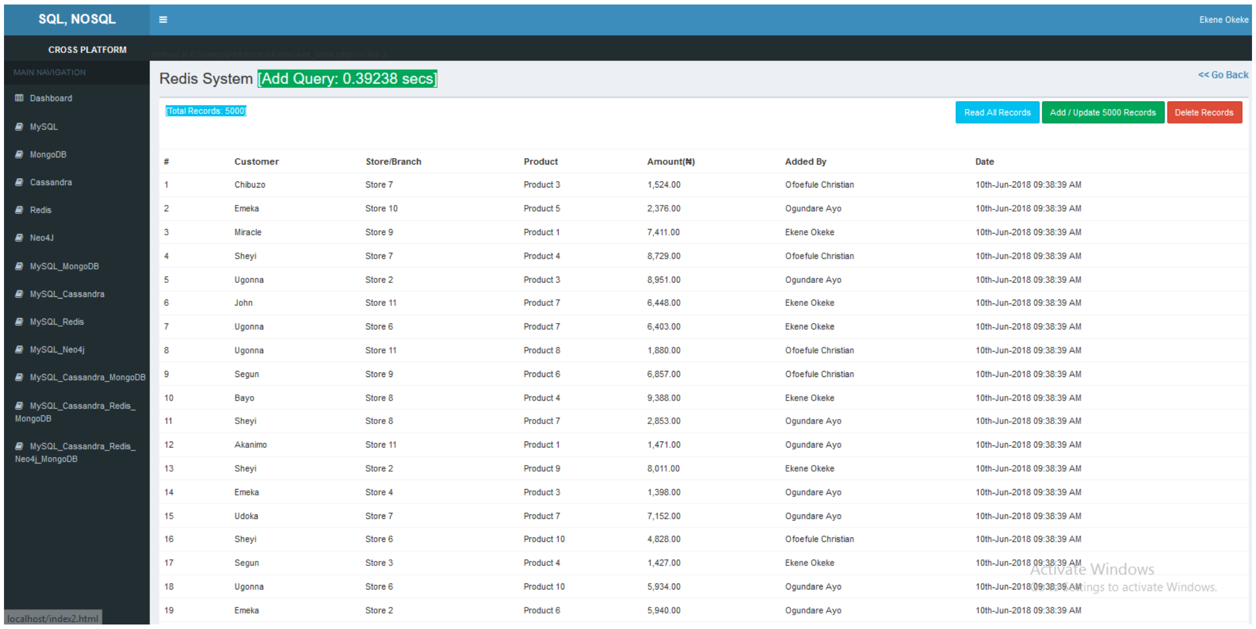
Redis System [Delete Query: 0.04501 secs]

[Total Records: 500]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Redis Delete Query for 500 records

Delete query was completed in 0.04501 seconds for Redis. The completion time shows fast delete operation for Redis.



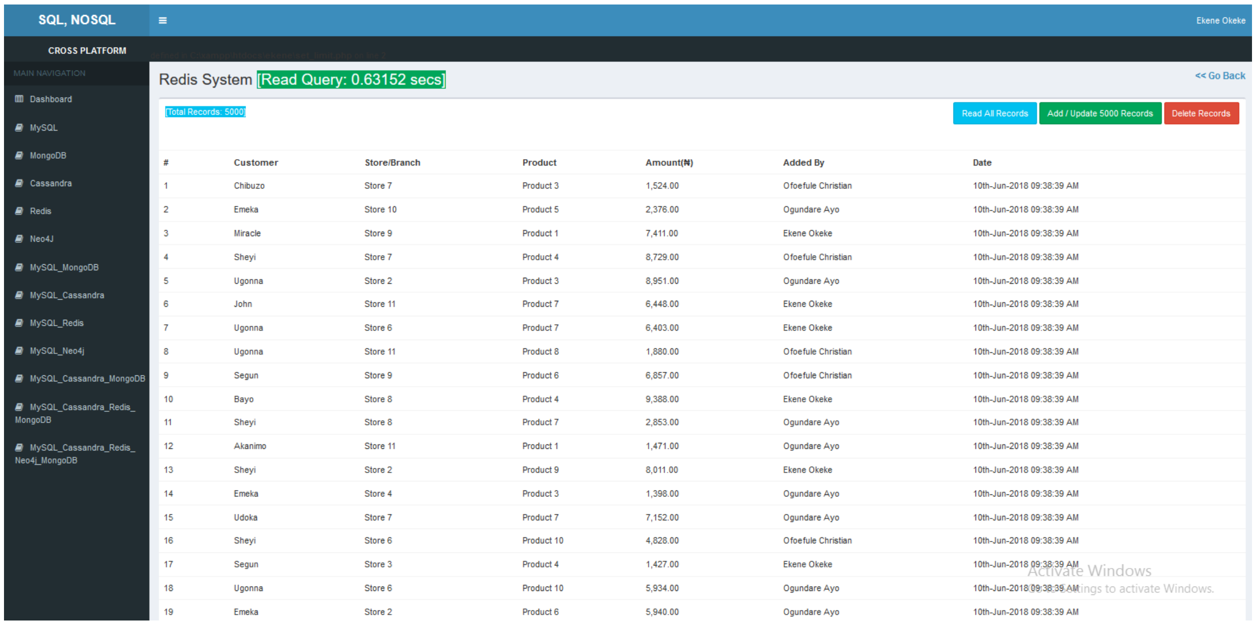
Redis System [Add Query: 0.39238 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Chibuzo	Store 7	Product 3	1,524.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
2	Emeka	Store 10	Product 5	2,376.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
3	Miracle	Store 9	Product 1	7,411.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
4	Sheyi	Store 7	Product 4	8,729.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
5	Ugonna	Store 2	Product 3	8,951.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
6	John	Store 11	Product 7	6,448.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
7	Ugonna	Store 6	Product 7	6,403.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
8	Ugonna	Store 11	Product 8	1,880.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
9	Segun	Store 9	Product 6	6,857.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
10	Bayo	Store 8	Product 4	9,388.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
11	Sheyi	Store 8	Product 7	2,853.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
12	Akanimo	Store 11	Product 1	1,471.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
13	Sheyi	Store 2	Product 9	8,011.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
14	Emeka	Store 4	Product 3	1,398.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
15	Udoka	Store 7	Product 7	7,152.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
16	Sheyi	Store 6	Product 10	4,828.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
17	Segun	Store 3	Product 4	1,427.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
18	Ugonna	Store 6	Product 10	5,934.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
19	Emeka	Store 2	Product 6	5,940.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM

Redis Insert Query for 5,000 records

Redis Insert query completed in 0.39238 seconds which is the fastest recorded time for insertion among the systems tested.



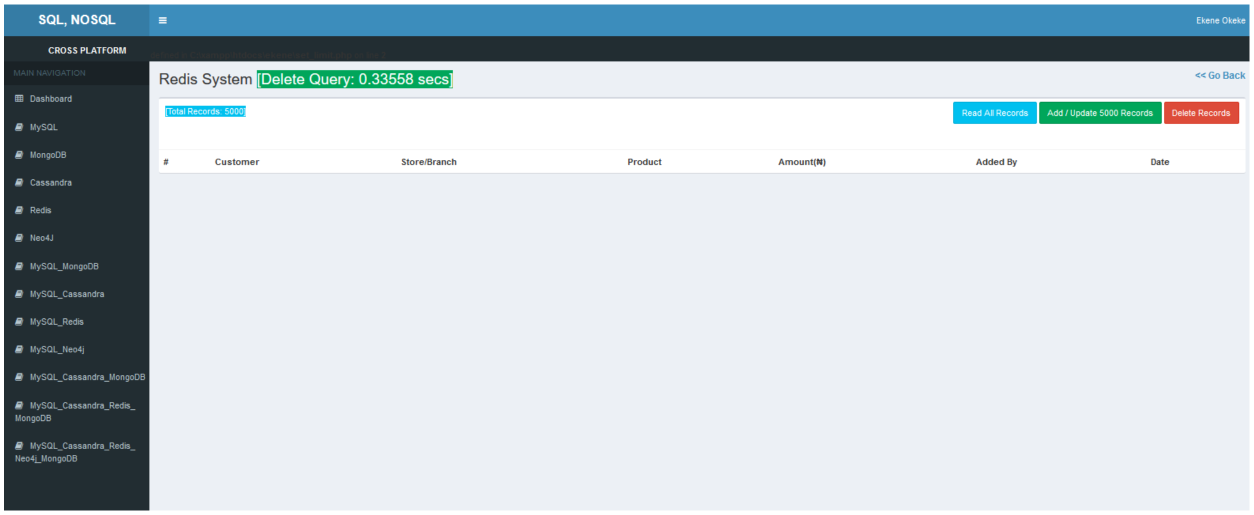
Redis System [Read Query: 0.63152 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Chibuzo	Store 7	Product 3	1,524.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
2	Emeka	Store 10	Product 5	2,376.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
3	Miracle	Store 9	Product 1	7,411.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
4	Sheyi	Store 7	Product 4	8,729.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
5	Ugonna	Store 2	Product 3	8,951.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
6	John	Store 11	Product 7	6,448.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
7	Ugonna	Store 6	Product 7	6,403.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
8	Ugonna	Store 11	Product 8	1,880.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
9	Segun	Store 9	Product 6	6,857.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
10	Bayo	Store 8	Product 4	9,388.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
11	Sheyi	Store 8	Product 7	2,853.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
12	Akanimo	Store 11	Product 1	1,471.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
13	Sheyi	Store 2	Product 9	8,011.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
14	Emeka	Store 4	Product 3	1,398.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
15	Udoka	Store 7	Product 7	7,152.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
16	Sheyi	Store 6	Product 10	4,828.00	Ofoefule Christian	10th-Jun-2018 09:38:39 AM
17	Segun	Store 3	Product 4	1,427.00	Ekene Okeke	10th-Jun-2018 09:38:39 AM
18	Ugonna	Store 6	Product 10	5,934.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM
19	Emeka	Store 2	Product 6	5,940.00	Ogundare Ayo	10th-Jun-2018 09:38:39 AM

Redis Read Query for 5,000 records

Redis read operation finished in 0.63152 seconds. Redis does not perform read-intensive operations better than MySQL and MongoDB as these two systems have been enhanced for read operations.



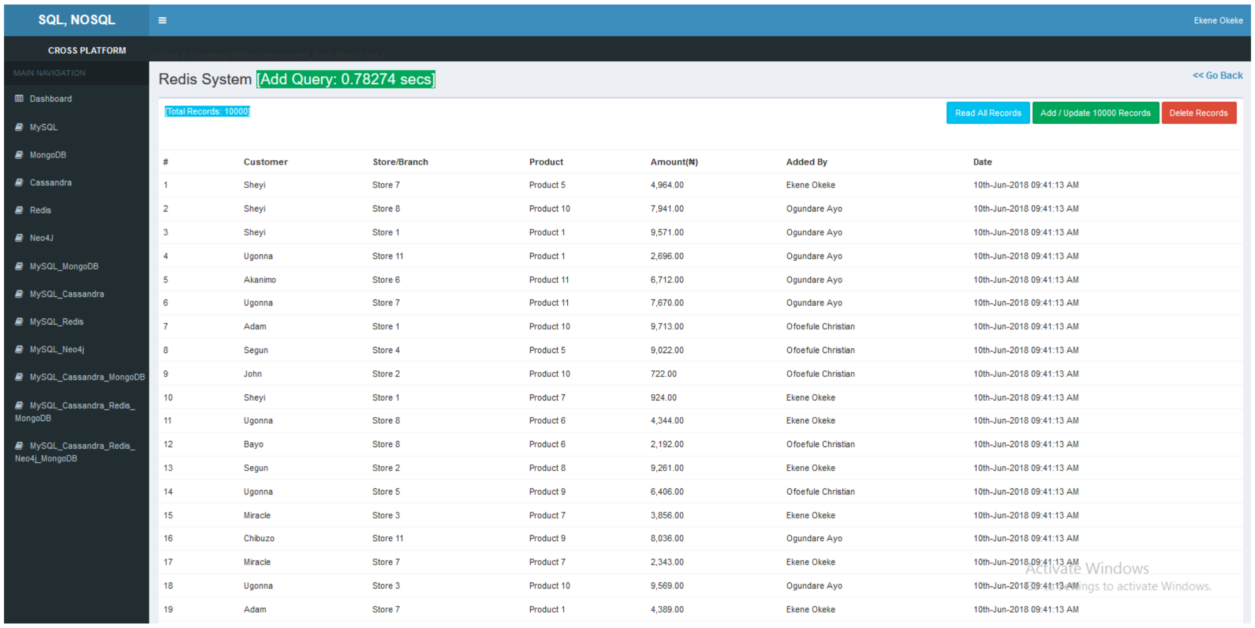
Redis System [Delete Query: 0.33558 secs]

[Total Records: 5000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Redis Delete Query for 5,000 records

Delete query for 5,000 records was completed in 0.33558 seconds in Redis system. The MongoDB and MySQL performed better than Redis in terms of delete operation with 0.12341 and 0.28454 respectively.



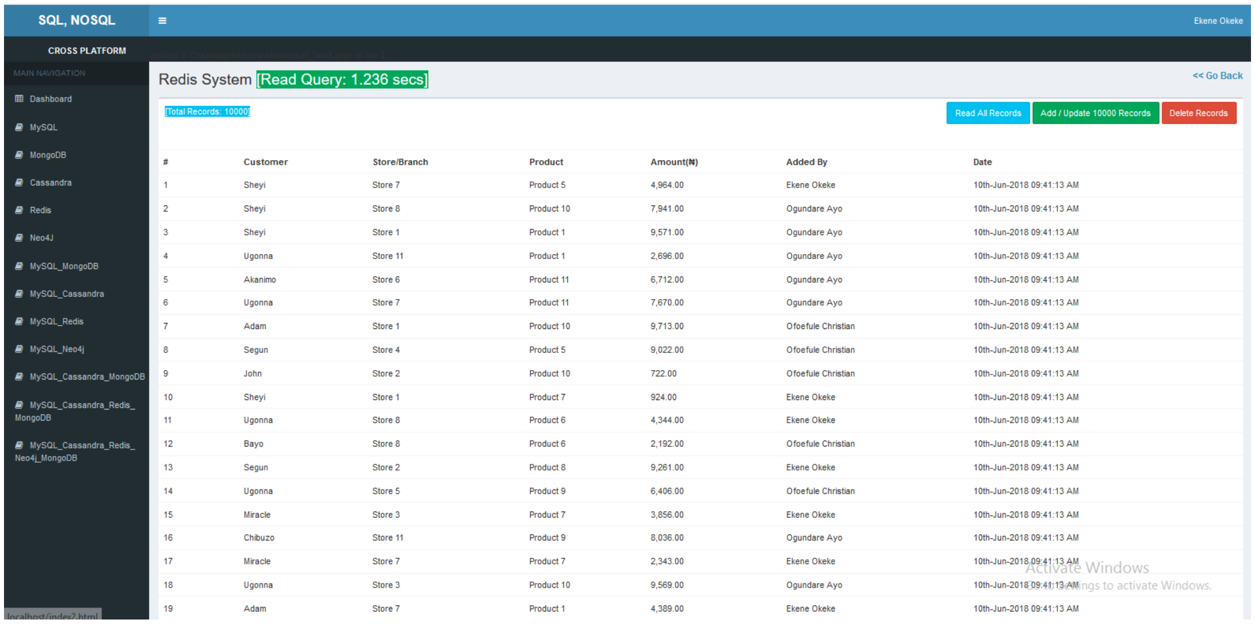
Redis System [Add Query: 0.78274 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 7	Product 5	4,964.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
2	Sheyi	Store 8	Product 10	7,941.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
3	Sheyi	Store 1	Product 1	9,571.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
4	Ugonna	Store 11	Product 1	2,696.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
5	Akanimo	Store 6	Product 11	6,712.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
6	Ugonna	Store 7	Product 11	7,670.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
7	Adam	Store 1	Product 10	9,713.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
8	Segun	Store 4	Product 5	9,022.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
9	John	Store 2	Product 10	722.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
10	Sheyi	Store 1	Product 7	924.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
11	Ugonna	Store 8	Product 6	4,344.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
12	Bayo	Store 8	Product 6	2,192.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
13	Segun	Store 2	Product 8	9,261.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
14	Ugonna	Store 5	Product 9	6,406.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
15	Miracle	Store 3	Product 7	3,856.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
16	Chibuzo	Store 11	Product 9	8,036.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
17	Miracle	Store 7	Product 7	2,343.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
18	Ugonna	Store 3	Product 10	9,569.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
19	Adam	Store 7	Product 1	4,389.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM

Redis Insert Query for 10,000 records

The insert query had a completion time of 0.78274 which is even faster than the MongoDB insert operation which was completed in 1.27148 seconds. Redis is the fastest system tested in terms of Add/ insert operations.



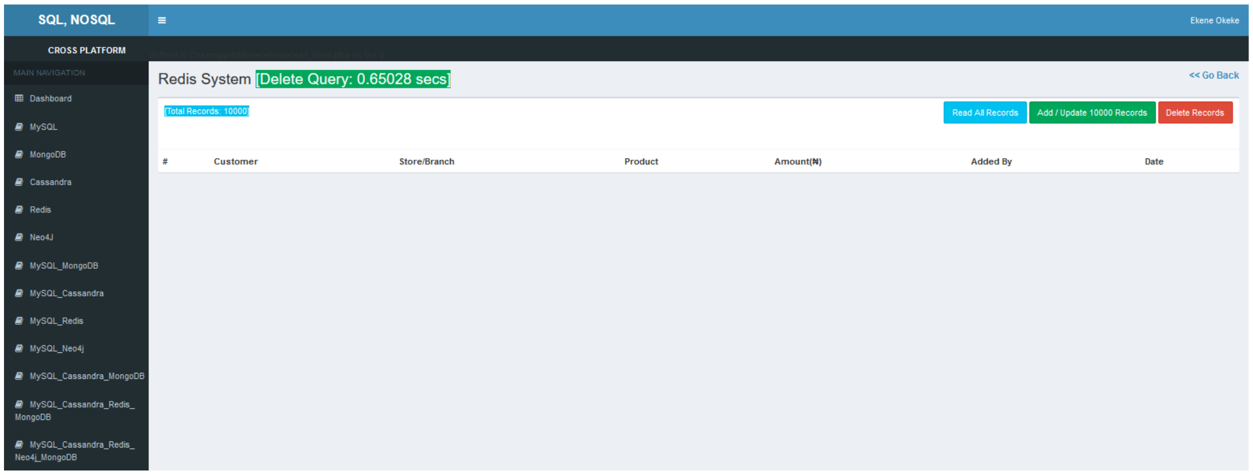
Redis System [Read Query: 1.236 secs]

[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Sheyi	Store 7	Product 5	4,964.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
2	Sheyi	Store 8	Product 10	7,941.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
3	Sheyi	Store 1	Product 1	9,571.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
4	Ugonna	Store 11	Product 1	2,696.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
5	Akanimo	Store 6	Product 11	6,712.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
6	Ugonna	Store 7	Product 11	7,670.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
7	Adam	Store 1	Product 10	9,713.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
8	Segun	Store 4	Product 5	9,022.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
9	John	Store 2	Product 10	722.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
10	Sheyi	Store 1	Product 7	924.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
11	Ugonna	Store 8	Product 6	4,344.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
12	Bayo	Store 8	Product 6	2,192.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
13	Segun	Store 2	Product 8	9,261.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
14	Ugonna	Store 5	Product 9	6,406.00	Ofoefule Christian	10th-Jun-2018 09:41:13 AM
15	Miracle	Store 3	Product 7	3,856.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
16	Chibuzo	Store 11	Product 9	8,036.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
17	Miracle	Store 7	Product 7	2,343.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM
18	Ugonna	Store 3	Product 10	9,569.00	Ogundare Ayo	10th-Jun-2018 09:41:13 AM
19	Adam	Store 7	Product 1	4,389.00	Ekene Okeke	10th-Jun-2018 09:41:13 AM

Redis Read Query for 10,000 records

Read operation was completed in 1.236 seconds depicting a relative fast read time. Read operations are not the strong point of the Redis system as they perform better in insertions.



Redis System [Delete Query: 0.65028 secs]

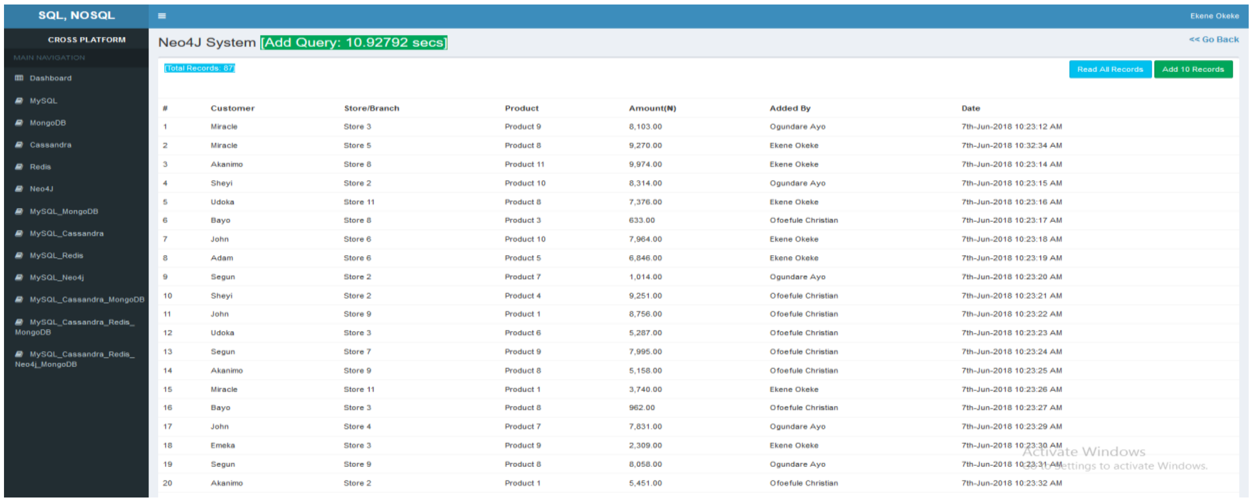
[Total Records: 10000]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date

Figure 4 Redis Delete Query for 10,000 records

The Redis System complted the delete query in o.65028 seconds which shows that Redis is fast for data deletion.

Neo4j System



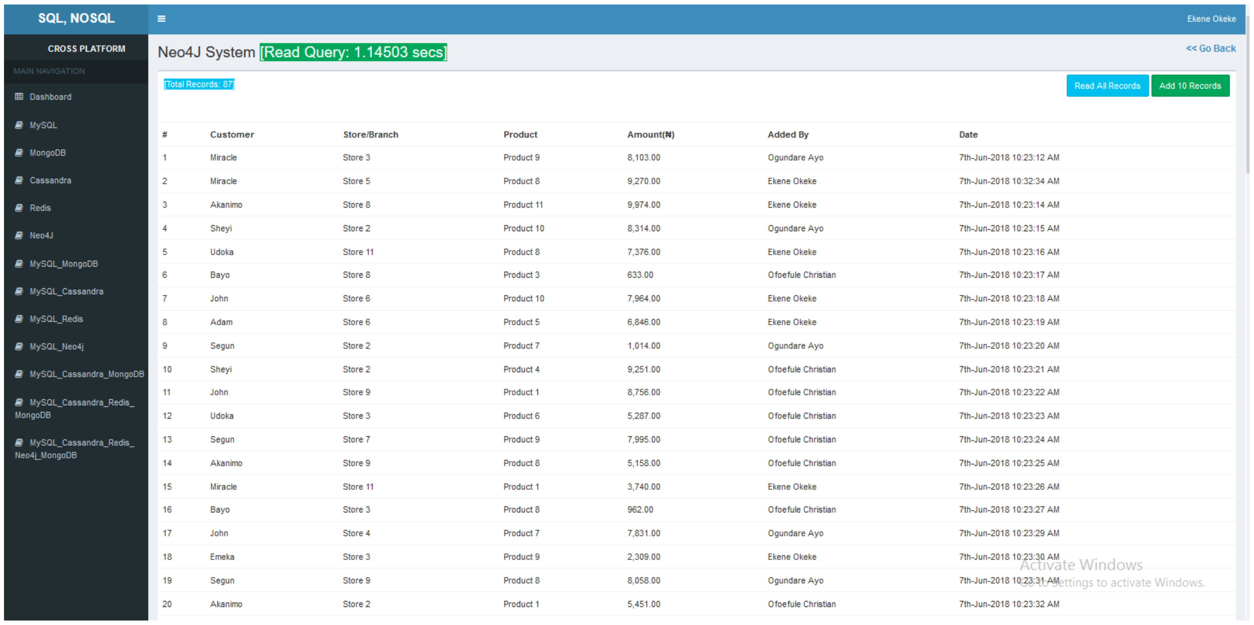
Neo4J System [Add Query: 10.92792 secs]

[Total Records: 87]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Insert Query for 10 records

The insertion of 10 records took a total of 11 seconds to accomplish which signifies slowness in creation of records for Noe4j system.



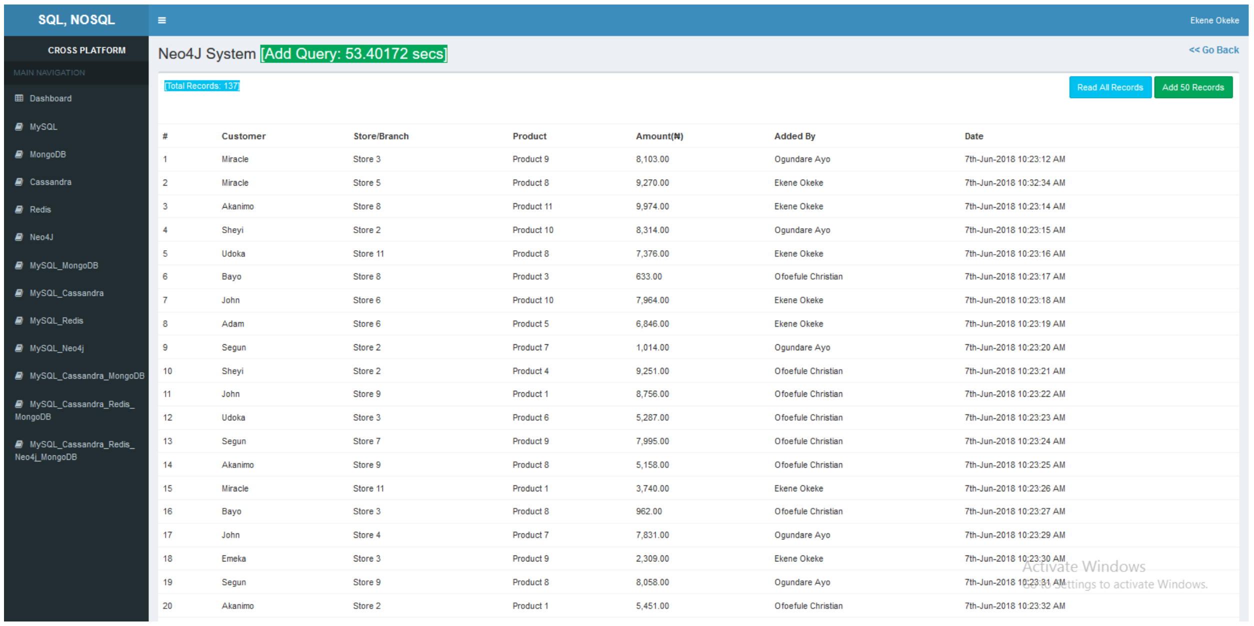
Neo4J System [Read Query: 1.14503 secs]

[Total Records: 87]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Read Query for 87 records

Read query for 87 records was completed in 1.145 seconds in the Neo4j system. Read operations are performed faster than insert operations in Neo4j graph-based NoSQL system.



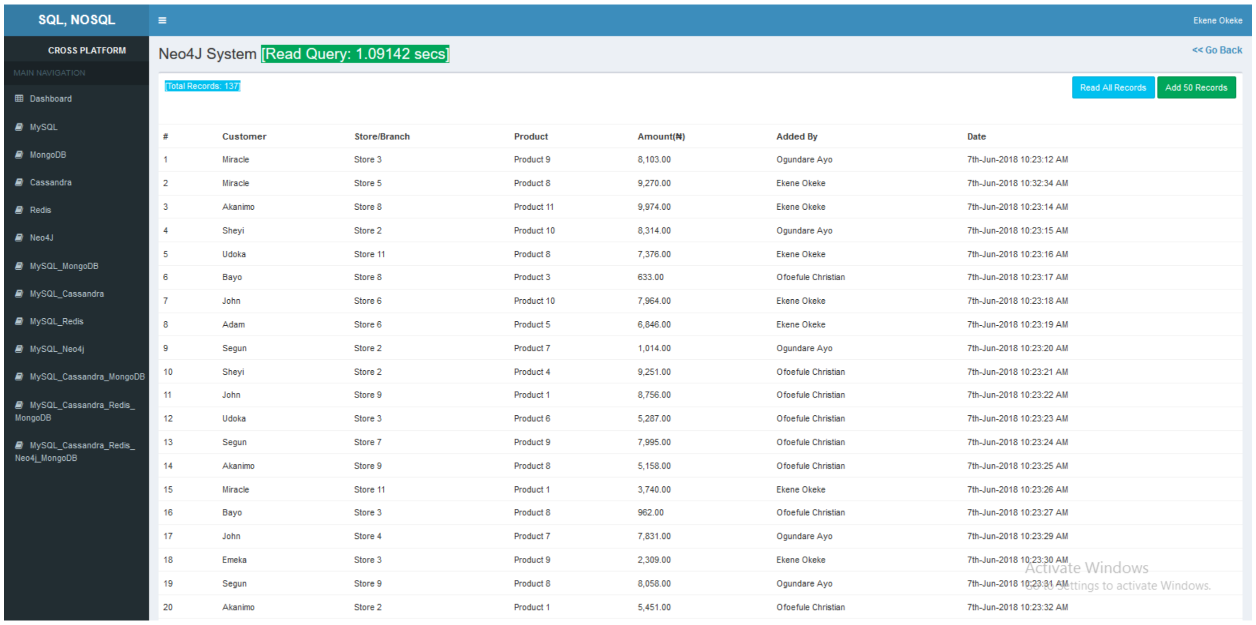
Neo4J System [Add Query: 53.40172 secs]

[Total Records: 137]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Insert Query for 50 records

Neo4j accomplished insert query of 50 records in 53.402 seconds which is approximately 1 minute. Insertions are performed slowly for query data in Neo4j.



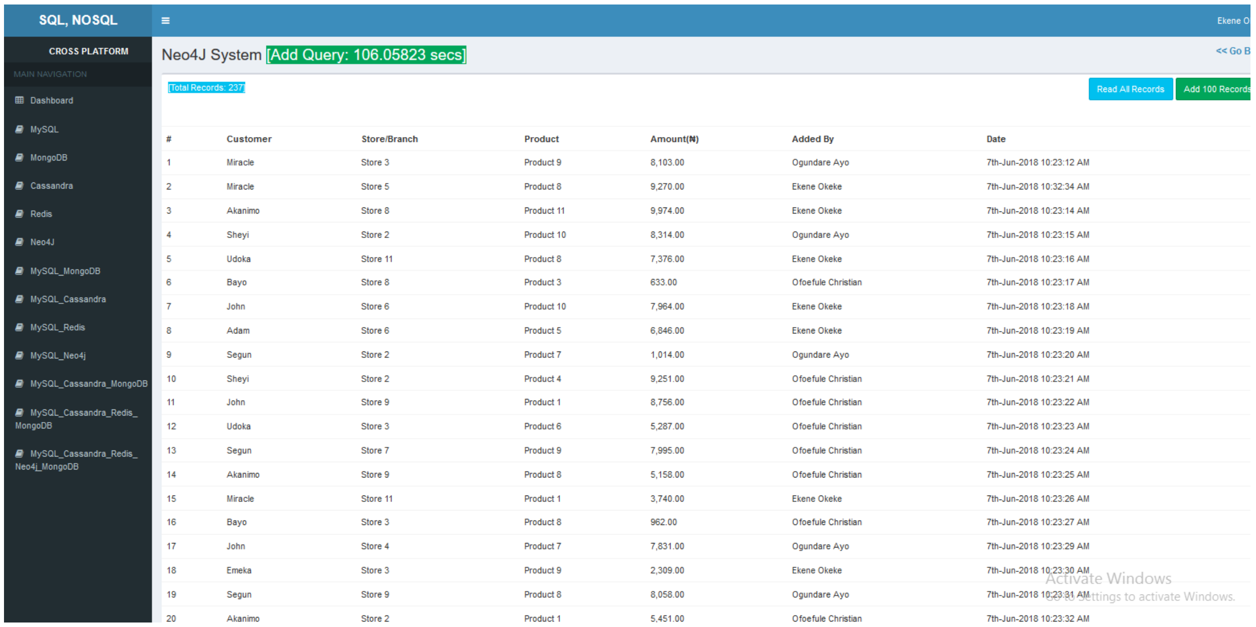
Neo4J System [Read Query: 1.09142 secs]

[Total Records: 137]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Read Query for 137 records

The Read operation for 137 records was completed in 1.091 seconds which indicates slowness in reading records by the Neo4j system.



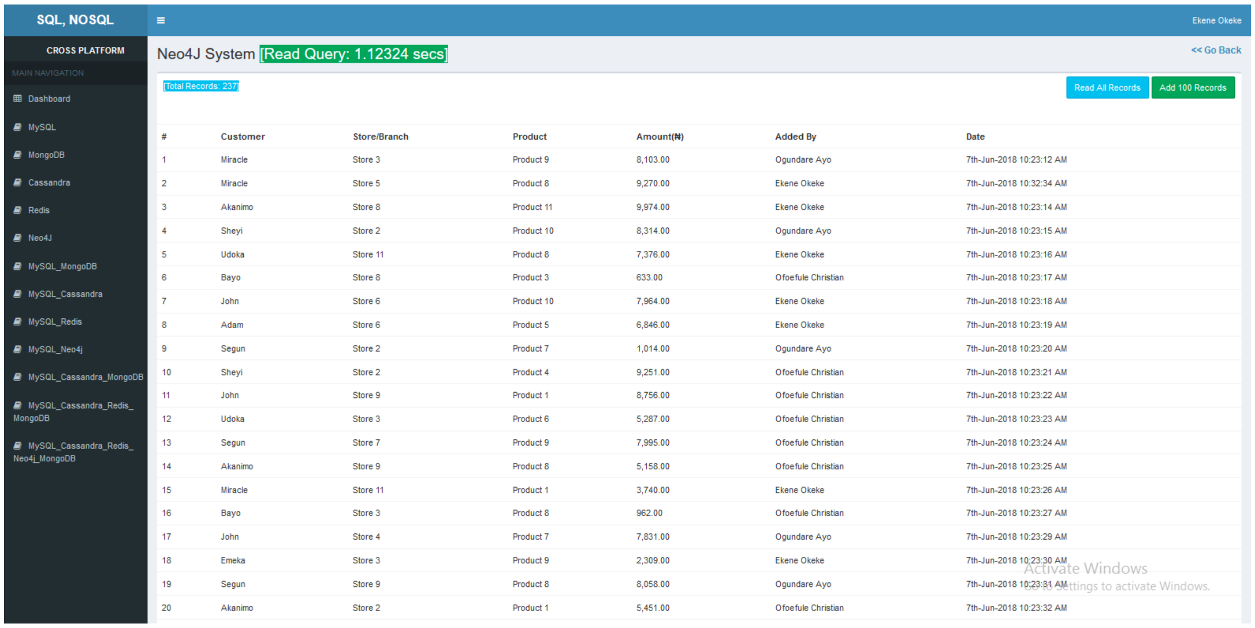
Neo4J System [Add Query: 106.05823 secs]

[Total Records: 237]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Insert Query for 100 records

The insert operation for 100 records was completed in 106.058 seconds. Small records were inserted in Neo4j as it does not have the capacity to accommodate vast volumes of query data.



Neo4J System [Read Query: 1.12324 secs]

[Total Records: 237]

#	Customer	Store/Branch	Product	Amount(₦)	Added By	Date
1	Miracle	Store 3	Product 9	8,103.00	Ogundare Ayo	7th-Jun-2018 10:23:12 AM
2	Miracle	Store 5	Product 8	9,270.00	Ekene Okeke	7th-Jun-2018 10:32:34 AM
3	Akanimo	Store 8	Product 11	9,974.00	Ekene Okeke	7th-Jun-2018 10:23:14 AM
4	Sheyi	Store 2	Product 10	8,314.00	Ogundare Ayo	7th-Jun-2018 10:23:15 AM
5	Udoka	Store 11	Product 8	7,376.00	Ekene Okeke	7th-Jun-2018 10:23:16 AM
6	Bayo	Store 8	Product 3	633.00	Ofoefule Christian	7th-Jun-2018 10:23:17 AM
7	John	Store 6	Product 10	7,964.00	Ekene Okeke	7th-Jun-2018 10:23:18 AM
8	Adam	Store 6	Product 5	6,846.00	Ekene Okeke	7th-Jun-2018 10:23:19 AM
9	Segun	Store 2	Product 7	1,014.00	Ogundare Ayo	7th-Jun-2018 10:23:20 AM
10	Sheyi	Store 2	Product 4	9,251.00	Ofoefule Christian	7th-Jun-2018 10:23:21 AM
11	John	Store 9	Product 1	8,756.00	Ofoefule Christian	7th-Jun-2018 10:23:22 AM
12	Udoka	Store 3	Product 6	5,287.00	Ofoefule Christian	7th-Jun-2018 10:23:23 AM
13	Segun	Store 7	Product 9	7,995.00	Ofoefule Christian	7th-Jun-2018 10:23:24 AM
14	Akanimo	Store 9	Product 8	5,158.00	Ofoefule Christian	7th-Jun-2018 10:23:25 AM
15	Miracle	Store 11	Product 1	3,740.00	Ekene Okeke	7th-Jun-2018 10:23:26 AM
16	Bayo	Store 3	Product 8	962.00	Ofoefule Christian	7th-Jun-2018 10:23:27 AM
17	John	Store 4	Product 7	7,831.00	Ogundare Ayo	7th-Jun-2018 10:23:29 AM
18	Emeka	Store 3	Product 9	2,309.00	Ekene Okeke	7th-Jun-2018 10:23:30 AM
19	Segun	Store 9	Product 8	8,058.00	Ogundare Ayo	7th-Jun-2018 10:23:31 AM
20	Akanimo	Store 2	Product 1	5,451.00	Ofoefule Christian	7th-Jun-2018 10:23:32 AM

Neo4j Read Query for 237 records

The Read operation for 237 records which is a small amount of records was completed in 1.123 seconds in Neo4j system. This signifies that Neo4j is slow in reading records.