بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

**Republic of Sudan**

**Shendi University**

**College of Post Graduate Studies and Scientific Research**

**Title:**

# NOSQL Data base Retrieval Using Hash algorithm for column store

***A thesis Submitted in Fulfillment of the Requirements of the Degree of M.sc in Computer Sciences***

**By:** Noah Omer Mohamed Ahmed

**Supervisor Assoc. Prof.:** Mohammed Bakri Bashir Elsafi

***May 2019***

# Abstract

The NoSQL Data Storage is a storage infrastructure designed specifically to store, manage and retrieve large amounts of unstructured data. The Column-Based is the one type of Nosql database that store the data in a column structured. The column-based database stores these data in one file which it produced a delay in handling the user's queries, because the database handling the file sequential. Additionally the Nosql DBMS's don't support primary key concepts. This research developed hash-base indexing algorithm that enhance search process and speed up user's query response time. Furthermore, provide researchers and framework that they can use to test their indexing algorithm. The hash algorithm developed by using Java language and connect with Apache HBase DBMS, which installed in Hadoop server and using Zookeeper tool. The experiments conduct to measure the response time of the proposed (Hash) algorithm and Indexed algorithm .the result show that the Hash algorithm enhanced the response time when it compared with Indexed algorithm. The Hash algorithm can be able to reduce the response time if it implements another algorithm to reduce collision of keys in the index table.

**المستخلص**

يعد تخزين بيانات NoSQL بنية أساسية للتخزين مصممة خصيصًا لتخزين وإدارة واسترداد كميات كبيرة من البيانات غير المهيكلة. المستندة إلى العمود هو نوع واحد من قاعدة بيانات Nosql التي تخزن البيانات في عمود منظم. تقوم قاعدة البيانات المستندة إلى العمود بتخزين هذه البيانات في ملف واحد مما أدى إلى تأخير في معالجة استفسارات المستخدم ، لأن قاعدة البيانات تتعامل مع الملف متسلسلة. بالإضافة إلى ذلك ، لا يدعم Nosql DBMS مفاهيم المفاتيح الأساسية. طور هذا البحث خوارزمية فهرسة ذات أساس تجزئة تعزز عملية البحث وتسريع وقت استجابة استعلام المستخدم. علاوة على ذلك ، وفر للباحثين إطار عمل يمكنهم استخدامه لاختبار خوارزمية الفهرسة الخاصة بهم. تم تطوير خوارزمية التجزئة باستخدام لغة Java والتواصل مع Apache HBase DBMS ، والتي تم تثبيتها في خادم Hadoop واستخدام أداة Zookeeper. تُجري التجارب لقياس زمن الاستجابة لخوارزمية (هاش) المقترحة وخوارزمية مفهرسة .وتوضح النتيجة أن خوارزمية هاش قد حسنت وقت الاستجابة عند مقارنتها بالخوارزمية المفهرسة. يمكن أن تكون خوارزمية التجزئة قادرة على تقليل وقت الاستجابة إذا نفذت خوارزمية أخرى لتقليل تصادم المفاتيح في جدول الفهرس.

# Acknowledgments

I'm greatly indebted to my supervisor **Mohammed Bakri Bashir Elsafi**, University of Shendi. Without him this work would have never been possible, and helped to solve all problems encountered throughout the entire duration of this research.

Thanks are also extended to the computer engineering of

**Abdel Fattah Awad Elkreem Ahmed** in Information Technology Center, Shensi University.

Many thanks extended to all members of the Faculty of Science & Technology, Department of Statistics & Computer Science.

# Dedication

I dedicate this work

To my mother

And my father

To all who gave efforts and facilitated this study

To my brother Mohammed.

To my sisters Rasha & Hanan for their continuous support and advice and all who have had positive impact on our lives.

To my friends for their support during my study.

To all of them I dedicate this work with real love and respect.

Noha.....

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| **Abbreviation** | **Description** |
|---|---|
| AP | Availability and Partition tolerance |
| APIs | Application Programming Interfaces |
| CAP theorem | Consistency, Availability, tolerance of network Partition |
| CP | Consistency and Partition |
| DB | Database |
| DBMS | Database Management Systems |
| DCODE | Distributed Column-Oriented Database Engine |
| DFS | Distributed File System |
| Disco | Distributed Co-clustering |
| DOT | Distributed Ordered Table |
| GFS | Google File System |
| HDFS | Hadoop Distributed File System |
| M2M | Multi-Column to Multi-Dimensional |
| M2S | multi-column to single dimensional |
| MBR | Minimum Bounding Rectangle |
| MDDS | Multidimensional Data Segment |
| MDRQ | Multi Dimensional Range Query |
| NoSQL | Not Only SQL |
| RDBMS | Relational Database Management Systems |
| S2S | Single column to Single dimensional |

# Chapter one:

## Introduction

## 1.1 Over view

New web applications like facebook and twitter produce large unstructured data that challenge mining, processing, modeling, security, indexing, analysis, storage, and Relation database[1] . Firstly, the size of the data is a larger than the reception and deal with the ordinary computers hosted on the file storage systems distributed (Distributed File System) (DFS) in the servers over the Internet. These storage systems ensure a low cost, access, and processing much faster than the storage in one unit. Secondly, the storage in the form of traditional methods such as a relational database is not suitable to store these data. Consequently, the applications such as Amazons and Facebook faced problems to store the received unstructured data in the relational database management systems (DBMS's). These challenges encouraged the researchers and companies to introduced new database system called Nosql.

NoSQL is a category of database that typically is schema free and does not follow relational model[2]. NoSQL is a collection of non-relational data, typically schema-free databases designed from scratch to address performance, scalability and availability issues of relational databases for storing structured or unstructured data. Consequently, these NoSQL databases classified into different categories by grouping the similar data in same category. Furthermore, these new databases are very different from traditional relational databases that use a SQL as query language[3], so it is referred to as "NoSQL" database. NoSQL also be interpreted as the abbreviation of "NOT ONLY SQL" to show the advantage of NoSQL.

## 1.2 Problem Background

With the development of internet application that uses Nosql DBMS and increasing database size which needs to be able to store and process effectively, the Nosql database systems facing many challenges. These challenges are various based on the Nosql database systems and the complexity of the data that stored in the DBMS's. Hence, the authors in [4]divided the data store model for Nosql into column-based, key value, graph, and document based model. The Column-Based is the one type of Nosql database that store the data in a column structured.

On one hand, the size of the data that produced by the web applications is unstructured and the column-based database stores these data in one file which it produced a delay in handling the user's queries, because the database handling the file sequential. Furthermore, the Nosql database does not support the primary key concepts, which affect the query process in the database file. On the other hand, the size of the data is very large as well as has a properties associated with it like volume and variability, but when facing increasing size which accused by delay. In addition, handling large data and storage systems used currently cannot store such large amount of data.

The aforementioned issues directed the research to find means to speed up the query processing. The addressing these issues either implementing an indexing algorithm or techniques during the storing the data in database, or by provide a query optimization techniques that reduce query response time.

## 1.3 Problem Statement

The unstructured nature of the Nosql data prevents the DBMS's in column base data base from implementing a primary key, which it's speeding up query execution. Consequently, defining a primary key or indexing is an important issue, which is used to reduce query processing time and increase system performance.

## 1.4 Research Objectives:

1. To study and review critically the indexing techniques in Nosql database and especially in Column stored DBMS's.
2. Design and implement a Nosql indexing algorithm for column-based database type.
3. Design and develop a framework to evaluate any indexing algorithm in NoSQL database.

## 1.5 Research Contributions:

This research is handling the response time delay by design a new indexing algorithm for Nosql database. Additionally, the indexing algorithms reduce the number of collisions in keys that has a numeric data type which appears during the indexing process. When applying hash functionality in Nosql, it improves response time, retrieval, and accessibility. Consequently, the contributions in this research are as following:

1- An Enhanced Hash indexing algorithm for column-based DBMS's.
2- A Framework to evaluate the NoSQL indexing algorithms.

## 1.6 Research Methodology:

The methodology presents to be adopted in continuing this research, which content the research procedures that Clearfield the focusing of this study, Problems Formulation ,which explain how the researcher find the Hash Indexing technique, Design and Development of hash indexing technique, Platforms and tools this included (Hadoop, Zookeeper, HBase) , dataset, technique design, experiment and evaluation are included.

## 1.7 Research Scope:

The research is focusing on NoSQL database query issues. The research is limited to column oriented models in the NoSQL. The index is hashing techniques in NoSQL.

- The proposed algorithm applied only in the column oriented Nosql databases only , while the other categories not used
- Security of Nosql is not concern in this research
- The algorithm developed to handle only text and numbers data because the column-store databases deal with text data.
- The operation used to test the algorithm is the retrieval which include search and retrieve the data.

## 1.8 Thesis Outline:

The remainder of this study is organized as follows.

**Chapter Two:** stared with a review for various indexing techniques using NoSQL, NoSQL database modeling,. These models are four types key value, column oriented, Document store and Graph store. There are column oriented store features and key primly index in data base. In addition to that Hash logarithms were used to predict index techniques in model column oriented store in NoSQL data base which used to measures accuracy rate of these techniques.

**Chapter Three:** covers the research methodology which includes research procedures, the operational framework, dataset, technique design, tools, platform, experiments test and evaluation.

**Chapter Four:** shows the designs of indexing techniques, beginning with preprocessing and what technique is suitable. Then, involves building indexes using NoSQL technique, presents the findings of this study and the discussion of the results, to evaluate technique support change in hash index introduced to predict index model which used to evaluate performance. In this experiment one against all approach.

The thesis concludes with **Chapter Five** which states the conclusion of the research. Additionally, it provides recommendations for future work.

# Chapter Two
# Literature review

## 2.1 Over view:

The increase of an unstructured Data volume during recent years produced issues for Traditional relational database management systems (RDBMS). RDBMS are facing challenges, such as high performance, huge storage, high scalability, and high availability. To handling these problems, a series of Not Only SQL (NoSQL) databases have been proposed and widely used in industry and academic fields such as Google Big Table, Yahoo! PNUTS, HBase and Cassandra. This chapter discuss in detail the Nosql concepts, which it started with an introduction to Nosql database management systems and its relation with the relation DBMS's. Then the chapter discuss the indexing techniques implemented in the literature that handling the problem of the primary key. The chapter concluded with comparative analysis of the indexing techniques in Nosql as well as the problem formulation.

## 2.2 Nosql database management systems concepts:

The large Data is set of Data with a large Volume, high Variety and complex structure, so the management, analyzing, storing and processing the data is very difficult [5]. Moreover the increase of the data produces a new type of data not recognized before. In addition to structure data and semi structured, an unstructured database is appear, which enforced the industries and the researchers to develop a new DBMS's. These New DBMS's which it's later called Nosql consider the features of unstructured DBMS's, which is different from the relation database management systems[6]. Additionally, the unstructured data is produced from web application such as Social media applications(facebook, twitter, skype) and so on generated data every second , which illustrated the size of the data and the nature of it as explained in Figure2.1.

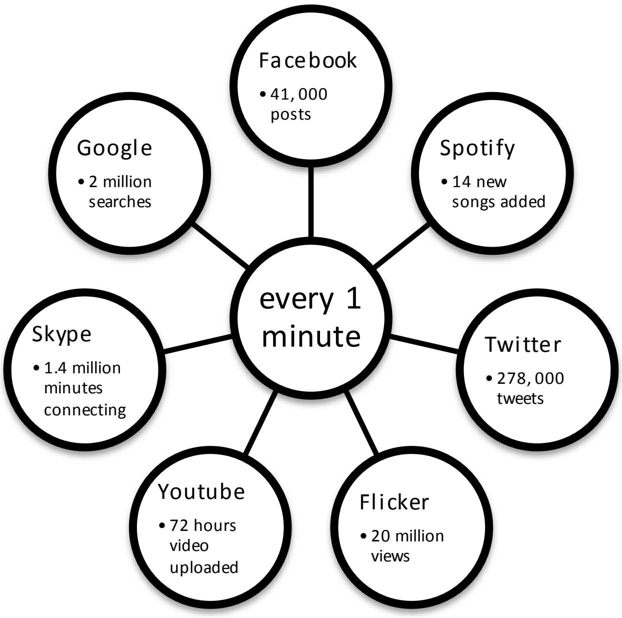


**Figure 2.1 A 60-second schematic of large data across social media[7]**

## 2.3 The relational and NOSQL databases:

Database Management Systems (DBMS) are the higher-level software, working with lower-level application programming interfaces (APIs) that take care of the database operations[1]. Furthermore, DBMSs have been developed for decades that provide operations to deal with data on the database such as processed, recorded, and retrieved operations. DBMS's can be divided into two main categories relational and non-relational or NoSQL databases. A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many ways without having to reorganize the database tables[8] . One of the relational database is SQL which is a database query language designed for the retrieval and management of data in a relational database.

The main Disadvantages of traditional database are[9]:

1. Limited capacity: Existing relational database cannot support large data in search engine, or Big System.

2. Expansion difficult: Multi-table correlation mechanism which exists in relational database became the major factor of database scalability.
3. Slow reading and writing: A relational database itself has a certain logic complexity, with the data size increases, it is prone to bring about deadlocks and other concurrency issues, this has led to the rapid decline in the efficiency of reading and writing[10].

On the other hand, the term “NoSQL” already exists since 1998. Carlo Stress named an open-source database “NoSQL” to make clear, that his project does not support any SQL interface. The underlying concept of his NoSQL-databases waives relations therefore the expression would be more appropriate. It is no common definition for NoSQL-databases available but Edict et.al. [11] .

Point out 7 important characteristics in NoSQL-databases:

1. Are not based on a relational approach.
2. Scale horizontal.
3. Are often open-source products (although proprietary products are available)
4. Don’t need a defined schema.
5. Provide an API for the integration in other software products.
6. Use a decentralized architecture for the easy replication of data.
7. Follow the BASE principle (Basically Available, Soft State, and eventually.

Meanwhile, NoSQL have some inadequacies, such as does not support SQL which is industry standard, lacking of transactions, reports and other additional features. Furthermore, NoSQL not mature enough for most of the NoSQL database products were created in recent years and so on[12] . Feature of NoSQL database described above are common ones, in reality, each product comply with the different data models and CAP theorem. Therefore, NoSQL database data model will introduce, and classify NoSQL according to CAP theorem[13].

The main advantages of NoSQL are[14]**:**

(1) High concurrent of reading and writing with low latency :

Database were demand to meet the needs of high concurrent of reading and writing with low latency, at the same time, in order to greatly enhance customer satisfaction, database were demand to help applications reacting quickly enough.

(2) Efficient large data storage and access requirements:

Large applications, such as search engines, need database to meet the efficient data storage and can respond to the needs of millions of traffic.

(3) High scalability and high availability

With the increasing number of concurrent requests and data, the database needs to be able to support easy expansion and upgrades, and ensure rapid uninterrupted service.

(4) Lower management and operational costs

With the dramatic increase in data, database costs, including hardware costs, software costs and operating costs, have increased. Therefore, need lower costs to store large data. Although relational databases have occupied a high position in the data storage area, but when facing above requirements, it has some inherent limitations.

To solve several needs above, a variety of new types of databases management systems appeared.

## 2.4MainstreamNoSQLData Models:

Data model of traditional database are mainly relational, specifically to support associated class operations and transactions. But in the NoSQL database fields according to their characteristics NoSQL-databases can be divided into four groups , the mainstream data model is the following:

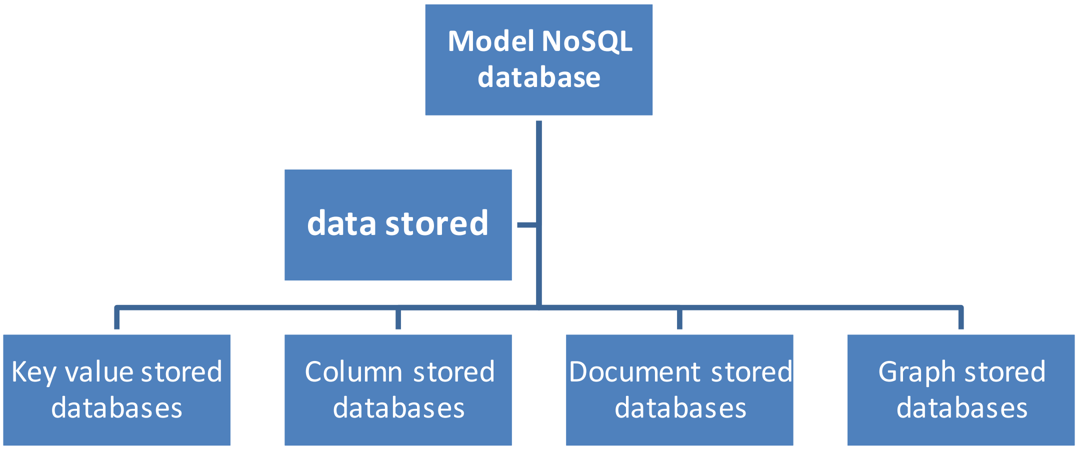


**Figure 2.2Model NoSQL[12]**

### 2.4.1 Key value stored databases:

This kind of NoSQL databases use a simple schema based on key value pairs. Key-value data model means that a value corresponds to a Key, although the structure is simpler, the query speed is higher than relational database, support mass storage and high concurrency, etc., query and modify operations for data through the primary key were supported well. Large number of key value NoSQL DBMS's developed such as Aero spike, Apache Ignite, ArangoDB, Berkeley DB, Couch base, Dynamo, Foundation DB, Infinity DB, Me cache DB, MUMPS, Oracle NoSQL Database, Orient DB, Redis, Riak, Sci DB, SDBM/Flat File.

### 2.4.2 Column stored databases:

Data is stored in columns instead of rows. Column stored database using Table as the data model, but does not support table association. Column stored database has the following characteristics [15]:

(1) Data is stored by column that is data stored separately for each column.

(2) Each column of data is the index of database.

(3) Only access the columns involving the queries result to reduce the I/O of system.

(4) Concurrent process queries, that is, each column treat by one process.

(5) There have the same type of data, similar characteristics and good compression ratio.

Overall, the advantage of this data model is more suitable application on aggregation and data warehouse. Although column-oriented database has not subverted the traditional stored by row, but in architecture with data compression, shared-nothing, massively parallel processing, column-oriented database can maintain high-performance of data analysis and business intelligence processing. Column oriented databases management system such as HBase, Yale University Hadoop DB, Face book’s Cassandra[1], Hypertable [16], Google's Big table[9], and Yahoo's PNUTS, Accumulo, Scylla, Apache Druid, Vertica.

#### 2.4.2. 1 HBase:

HBase is an open source project that provides transactional and indexing extension for HBase; Apache HBase is used to have random, real time read/write access to Big Data. It hosts very large tables on top of clusters of commodity hardware[17].

Apache HBase is a non relational database modeled after Google's big table, Big table acts up on Google File System; likewise Apache HBase works on top of Hadoop and HDFS. HBase is used whenever we need to provide fast random access to available data. Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

### 2.4.3 Document stored databases:

Data is not stored in tables but in documents. Documents refer to structured files, like JSON, YAML or RDF. Document database and Key-value is very similar in structure, but the Value of document database is semantic, and is stored in JSON or XML format. In addition, the document databases can generally a Secondary Index to value to facilitate the upper application, but Key-value database cannot support this.

Document database is not concerned about high- performance read and write concurrent, but rather to ensure that big data storage and good query performance.  Typical document database, Mongo DB[18], Couch DB[19] , ArangoDB, BaseX, Cluster point.

### 2.4.4 Graph stored databases:

Data is stored as graph or tree structures which link the different data aspects. Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties[20]. In graph databases, traversing the joins or relationships is very fast. The relationship between nodes is not calculated at query time but is persisted as a relationship. Traversing persisted relationships is faster than calculating them for every query. Some of the popular graph databases are Infinite Graph, Flock DB , Allegro Graph, ArangoDB, Apache Graph, Mark Logic, Neo4J, Orient DB, Virtuoso.

## 2.5 CAP theorem and NoSQL database classification:

In 2000, Professor Eric Brewer put forward the famous CAP theorem.  That is, Consistency, Availability, tolerance of network Partition.  CAP theorem's core idea is a distributed system cannot meet the three district needs simultaneously, but can only meet two.

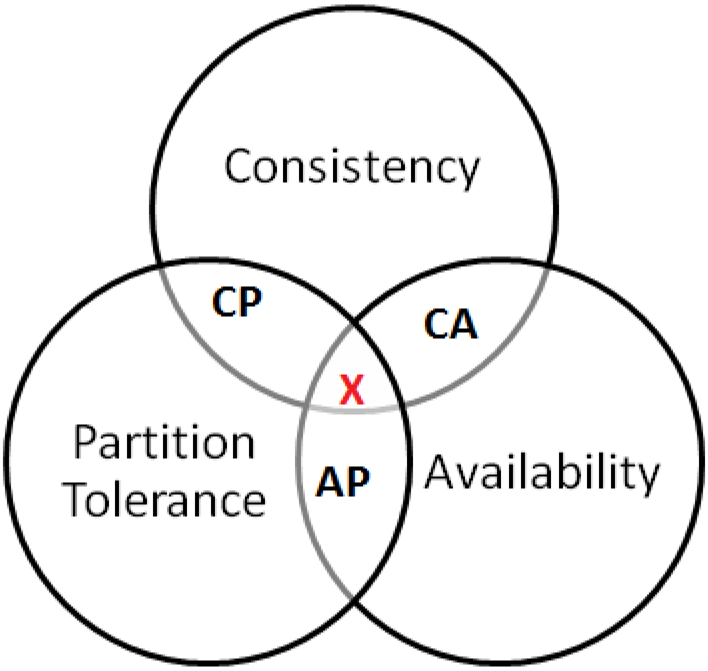


**Figure 2.3 CAP theorem[10]**

According to CAP theorem and different concerns of NoSQL database, a preliminary classification of NOSQL databases is as follows :

**• Concerned about consistency and availability (CA)**

Part of the database is not concerned about the partition tolerance [23], and mainly use of Replication approach to ensure data consistency and availability. Systems concern the CA are: the traditional relational database, Vertical (Column-oriented), Aster Data (Relational),Green plum (Relational) and so on.

**• Concerned about consistency and partition tolerance (CP):**

Such a database system stores data in the distributed nodes[21], but also ensure the consistency of these data, but support not good enough for the availability. The main CP system: Big Table (Column-oriented), Hyper table (Column oriented), HBase (Column oriented) Mongo DB (Document), Territory (Document), Red is (Key value), Scalar is (Key value) , Me cache DB (Key value), Berkeley DB (Key value).

**• Concerned about availability and partition tolerance (AP):**

Such systems ensure availability and partition tolerance primarily by achieving consistency. Such systems ensure availability and partition tolerance primarily by achieving consistency, AP's system: Volde mort (Key value), Tokyo Cabinet (Key value), KAI (Key value), Couch DB(Document-oriented),Simple DB(Document-oriented), Riak (Document-oriented).

## 2.6 Column family stores:

Column family databases store data in column families as rows that have many columns associated with a row key as shown in Figure 2.4Column families are groups of related data that is often accessed together. Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows. Some of the popular column family stores are Cassandra and HBase.

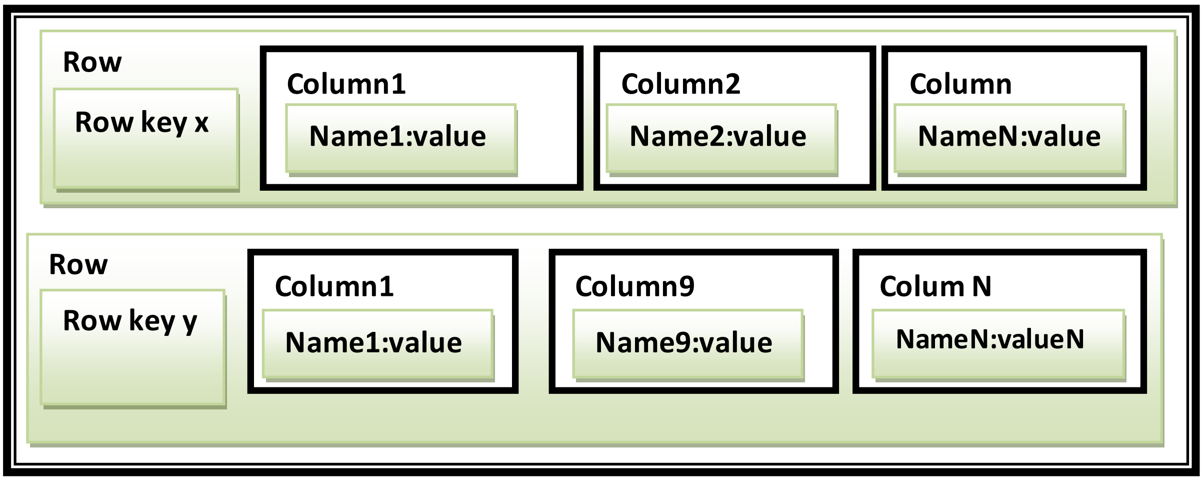


Figure 2.4 Column Family Stores[22]

## 2.7 Percussion of Column Family Stores:

### 2.7.1 Distributed Co-clustering:

The Distributed Co-clustering (Disco) framework introduces practical approaches for distributed data pre-processing, and co-clustering. Disco developed using Hadoop, an open source Map-Reduce implementation. The experiments show that Disco can scale well and efficiently process and analyze extremely large datasets (up to several hundreds of gigabytes) on commodity hardware[23].

### 2.7.2 A Distributed Column-Oriented Database:

Distributed Column-Oriented Database Engine (DCODE) for ancient analytic query processing that combines advantages of both column storage and parallel processing. DCODE enhance an existing open-source columnar database engine by adding the capability for handling queries over a cluster[24].

### 2.7.3 Distributed Ordered Table in NoSQL:

The way of storing the data in Nosql DBMS is affecting the process of the data and the query execution time. The need for design and develop a new technique to handle the delay in the query process is an important issue[25]. A lot of Not Only SQL (NoSQL) databases have been proposed in the era of large data. that has attracted lots of attention[26]. Distributed Ordered Table (DOT) horizontally partitions table into regions and distributes regions to region servers according to the keys. Multi-Dimensional Range Query (MDRQ) is a common operation over DOTs. Many indexing techniques have been proposed to improve the performance of MDRQ, but they cannot guarantee high performance[27].

### 2.7.3.1 Procedure of Distributed Ordered Table in NoSQL:

**A. Multidimensional Data Segment (MDDS):**

A data segment includes the following sections. First, the header stores the segment's unique identifier, its total length in bytes, and the record format's (the same as in the record descriptor). Second, the indexing dimensions section indicates the record fields used for data indexing, as in the record descriptor. Lastly, the records section stores the sequence of data records as an array of bytes. These sections, except the header, are of variable length, which is stored in their section length field[28].

**B. Record Descriptor**

MDDS requires the user to provide record descriptor, in XML format, with the record structure of the ingress data[29] .

**C. Software Architecture:**

MDDS's main components have been written C++ language and its architecture is illustrated.

**Data Investor:** The data investor produces data segments from every chunk of records it receives. To do so, it performs two operations on each chunk: data segmentation and segment assembling.

**Indexing and Writing Segments to Storage:** Right after being assembled, a data segment is in memory, in the right format to be accessed by the query processor, and ready to be written to storage.

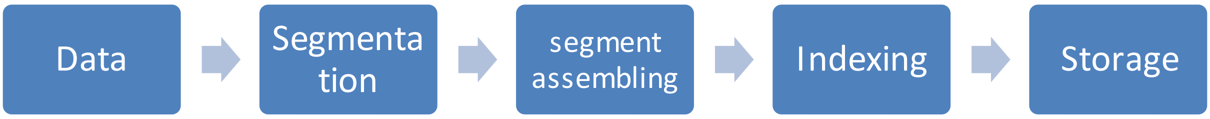


Figure 2.5 MDDS Architecture

## 2.8 Indexing of NoSQL in Column Family Stores:

Unnecessary full table scans cause a huge amount of unnecessary I/O and can drag down an entire database. The tuning expert first evaluates the SQL based on the number of rows returned by the query. The most common tuning remedy for unnecessary full-table scans is adding indexes. The Primary Key for a table acts as a default index. Additional indexes can be added to a table depending upon the data size it holds other type of indexes like Standard b-tree indexes and Bitmapped and function-based indexes can also eliminate full-table scans.

### 2.8. 1 Mapping between Indexed Columns and the Row key of Index Table:

Index data can be stored by the table of DOTs or other data structures. Each data structure must have a unique identifier to identify the index data. For tables of DOTs, the identifier is the row key for R-tree, it is the ranges of Minimum Bounding Rectangle (MBR). The unique identifier is called row key, because most indexing techniques use tables of DOTs to store indexes. Similarly, the data structures storing index data are called index tables. The tables of DOTs are single dimensional. DOTs identify records by the row key for insert and query operations. The row key of an index table may contain one or more indexed columns. Based on the mapping between indexed columns (source) and the row key of index table (destination), there are three kinds of mappings: single column to single dimensional, multi-column to multi-dimensional, and multi-column to single dimensional. These mappings determine how the indexes are built and how queries are processed.

#### 2.8. 1.1 Single Column to Single Dimensional:

A single column to single dimensional (S2S) index builds one index table for each indexed column and usually stores index data in tables of DOTs. The row key of an index table is usually concatenated by the value of indexed column and the raw row key. The S2S index is simple to implement, and its drawback is that only one indexed column can be altered by the row key of an index table.

#### 2.8.1.2 Multi-Column to Multi-Dimensional:

Spatial data structures like R-tree and R+-tree [15] have been widely used in spatial databases to store and query geometric objects. These structures map multi- dimensional data into multi-dimensional spaces. Multi- column to multi-dimensional (M2M) indexes use spatial data structures to map multiple indexed columns to the row key.

#### 2.8.1.3 Multi-Column to Single Dimensional:

A multi-column to single dimensional (M2S) index maps data of multiple indexed columns into the row key of an index table. The most common mapping function used is the space filling curve like Order curve and Hilbert curve. The advantages of M2S indexes are that they can use the table structure of DOTs to store index data and can filter multiple indexed columns at the same time. However, using the curves alone can result in false positive sub queries.

### 2.8.2 Index Structure of Column Family Stores:

According to Silberschatz et al the index structure can be categories into secondary, or clustering, which is also applicative for indexing DOTs.

#### 2.8.2.1 Secondary indexes:

Secondary indexes provide mapping from the values of indexed columns to the row key of the raw table. Queries will be processed by firstly finding rows satisfying all conditions (whose row keys are called the final row keys) and then querying the raw table based on the fial row keys.

#### 2.8.2.2 Clustering:

Clustering indexes have the same row key as secondary indexes and store all target columns of queries; therefore results can be directly returned without querying the raw table.

### 2.8.3 Time to Build Indexes:

DOTs usually use LSM tree on regions to reduce insert latency, namely, a record will be firstly inserted into the MemStore. The MemStore will be flushed to disk when meeting some conditions, like reaching the threshold. Therefore there are two kinds of indexes, on insert or on flush.

#### 2.8.3.1 on Insert Index:

On insert indexes build indexes when records are inserted, and they can be on either client side or server side. Client side means indexes are built when records are inserted into the raw table. It leaves the responsibility of building and maintaining indexes to users.

#### 2.8.3.2 on Flush Index:

On flush indexes will not build indexes for in- memory records until they are flushed to the disk. Inserts return directly without building indexes, which significantly reduces the insert latency. However, a query must be processed on both indexed data on disk and non-indexed data in MemStore.

## 2.9 Index Persistence:

As the name indicates, DOTs are distributed systems. No matter where the index data is stored, its reliability should always be considered carefully.

### 2.9.1 Persistent on Disk:

It is not necessary for users to worry about reliability when the index data is persistent on disk because the underlying distributed file systems like Google file systems (GFS) and Hadoop distributed file systems (HDFS) naturally provide transparent persistence. The disadvantage of persistent on disk index is that the storage cost is high. Using erasure codes to save index data is also considerable, but it leaves the risk of lower query performance when recovering data.

### 2.9.2 Non persistent on Disk:

To reduce the storage overhead, some techniques like CCIndex store only one copy of index data, which is called non persistent on disk.

### 2.9.3 Memory Only:

Memory only indexes store index in memory only for high performance of both insert and query. Indexes will be rebuilt when a region is migrated or the node is restarted.

## 2.10 A review on data index technique:

These reviews studied the research conducted to design and develop an index technique for NoSQL database. This study focus in the column oriented database.

Gugnani, et al. [30] Proposed a complementally clustering index technique called CCIndex to handle the storing of the ever growing big data. The research uses the HBase NoSQL DBMS to test proposed technique which focuses only on columns-based type. The proposed technique is designs to accelerate index on HBase.

The CCIndex technique was developed by Zou, et al. [31]is a complementally clustering index on Distributed Ordered Tables for accelerating multi-dimensional range queries built on Apache HBase. CCIndex builds Complemented Clustering Index Tables (CCITs) for each index column. The index tables are regular HBase tables and are split into regions and stored on Region Servers. Each CCIT contains data for all columns. Thus, ranges queries can be evaluated using a simple scan operation and involve no random reads. CCIndex uses the region-to-server mapping information provided by HBase meta-tables to estimate the result size of queries and optimize the query plan.

Ren, et al. [32] introduce a middleware called IndexFS that adds support to existing file systems such as, Luster, and HDFS for scalable high-performance operations on metadata and small files. IndexFS uses a table-based architecture that incrementally partitions the namespace on a per-directory basis, preserving server and disk locality for small directories.

Feng, et al. [33] developed indexing technique named LCindex, short for Local and Clustering Index, to solve this issue. Experimental results confirm that LCIndex can achieve high performance on both insert operations and flexible MDRQ. LCIndex techniques have been proposed to improve the performance of MDRQ, but they cannot guarantee high performance on both insert and flexible MDRQ at the same time.

Wang, et al. [34] proposed an adaptive indexing technique to speed up a complex data query on HBase for IoT based SG big data a indexing techniques for complex queries over the SG dataset to efficiently exploit the rich connotations of data to enable characteristic analytics and fault prediction. As part of popular big data platform, HBase is replacing classic relational databases to host huge heterogeneous data records in the form of key-value storage. However, most existing secondary index schemes on HBase are managed and retrieved by corresponding data columns instead of queries to incur inefficiency in answering a complex data query.

Guo, et al. [35] the ICF-HBase index method has great significance on large scale data query of HBase. The HBase database only supports the row key index method, and index could not be constructed on a non row key field. To solve this problem, this paper proposes a general HBase secondary index method based on Isomorphic column family(ICF-HBase), which construct key-value index in an index family of the original data table, in order to realize the single key index, multi key index, data import and optimized query mechanism However, there are still some problems in HBase, such as the query pattern that supports only the row key index, the lack of support for multi dimensional complex  query,  and the lack of support for cross transaction. However, these methods also bring some additional problems, such as the impact of the original region load balancing, data and index consistency and so on.

Qi, et al. [36] Proposed Consistency Analysis of Secondary Index on Distributed Ordered Tables; however, the DOT does not support queries very well other than the primary key. One solution to this problem is indexing. Many indexing techniques are focusing on how to improve the query ability, but do not care about the consistency between the index table and base data table. This paper focuses on the relationship between the consistency and the performance about the indexing techniques. This paper gives the dentition about the consistency between the index table and the base data table, and presents the inconsistency window to measure the degree of consistency.

Gao and Qiu [37] Proposed a Supporting Queries and Analyses of Large-Scale Social Media Data with Customizable and Scalable Indexing Techniques over NoSQL Databases, However, the level of indexing support varies significantly across different NoSQL databases, and their current index structures are not flexible enough to handle the queries in social media data analyses. To solve this problem, a general customizable indexing framework proposed that can be built over distributed NoSQL databases. This framework allows users to define customized index structures that contain the exact necessary information about the original data so as to achieve efficient queries about social events.

[38]Secondary index is another kind of index techniques for cloud data management. CCIndex is another kind of secondary index solutions based on Key-value store proposed by in one secondary index table was built for each indexed column. In order to reduce the random read, the detailed information of each record was pushed into the secondary index table, so that the random read can be changed into sequential read. The author also proposed some optimization methods to support multi-dimensional query. CCIndex is easily to be implemented, but it has several drawbacks.

Wu, et al. [39] Proposed B-tree Based Indexing for Cloud Data Processing. The research present a novel, scalable B+-tree based indexing scheme for efficient data processing in the Cloud. This paper presents CG-index (Cloud Global index), a secondary indexing scheme for Cloud storage systems. CG-index is designed for Cloud platform and built from scratch. It is tailored for online queries and maintained in an incremental way. It shares many implementation strategies with shared-nothing databases , peer- to-peer computing , and existing Cloud storage systems CG-index supports usual dictionary operations (insert, delete and lookup), as well as range search with a given key range.

Liang and Yang [40] Provided several techniques, e.g., MD-HBase to help researchers have good understanding on current multidimensional indexing techniques and their design of new multidimensional indexing approach. Conduct extensive experimental study to learn the intrinsic characteristics of MD-HBase techniques. Results show that it is possible to build a multidimensional cloud indexing system that is both elastically scalable and efficient.

[Colmenares, et al. [41]] [42] proposed an ingestion, Indexing and Retrieval of High Velocity Multidimensional Sensor Data on a Single Node Its design centers around a two-level indexing structure, wherein the global index is an in-memory R*-tree and the local indices are serialized kd-trees. This study is confined to records with numerical indexing fields and range queries, and covers ingest throughput, query response time, and storage footprint. In addition, author evaluate a kd-tree partitioning based scheme for grouping incoming streamed data records. Compared to a random scheme.[Colmenares, et al. [42]][Colmenares, Dorrigiv [42]][Colmenares, Dorrigiv [42]][Colmenares, et al. [42]][Colmenares, Dorrigiv [42]][Colmenares, et al. [42]][Colmenares, Dorrigiv [42]][Colmenares, et al. [42]][Colmenares, Dorrigiv [42]][Colmenares, et al. [42]]

## Table 2.1 review index Technique of Nosql

| No | Paper Title | Year | Data Type | Application Area | Problem | Indexing Types | Indexing technique | Dataset (Size) | NOSQL Data base |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Characterizing and Accelerating Indexing Techniques on Distributed Ordered Tables | 2017 | Columns | | The problem of storing this ever growing | a complemental clustering index | CCIndex | Big size | HBase |
| 2 | CCIndex: a Complemental Clustering Index on Distributed Ordered Tables for Multi-dimensional Range Queries | 2017 | Columns | Text | High performance, low space overhead, and high reliability | a Complemental Clustering Index | CCIndex | Big size | HBase |
| 3 | IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion | | Columns | Table-based (Metadata). | important performance bottleneck for many distributed file systems in both the data intensive scalable computing (DISC) world and the high performance computing (HPC) world | file systems | IndexFS | largest file size(Terabytes) | HBase |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | LC Index: A Local and Clustering Index on Distributed Ordered Tables for Flexible Multi-Dimensional Range Queries | 2015 | Columns | Statistic | the performance of spatial trees relies on balanced data whereas the data distribution of indexed columns is usually unbalanced (e.g. limited traffic speed in a jam), which limits the flexibility of MDRQ of these indexing techniques | Local and Clustering Index | LC index | Big size | HBase |
| 5 | A Query-oriented Adaptive Indexing Technique for Smart Grid Big Data Analytics | 2017 | Columns | IoT (Internet of Things) based Smart Grid (SG) | Delay query retrieval time because there is no indexing | *Structure* | adaptive indexing technique | Big Data Analytics | HBase |
| 6 | A HBase Secondary Index Method Based on Isomorphic Column Family | 2017 | Columns | The HBase database | The HBase database only supports the rowkey index method, and index could not be constructed on a non rowkey field.(improve the retrieval performance of HBase by optimizing the structure of HBase schema and the design of rowkey) | isomorphic column family (ICF-HBase) Secondary Index | Rowkey | Big Data | -HBase |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | The Consistency Analysis of Secondary Index on Distributed Ordered Tables | 2017 | Columns | Social network, computer simulation, scientific computing | the DOT cannot support range query over non-key columns | Secondary Index | Secondary Index | Big Data | -HBase |
| 8 | Supporting Queries and Analyses of Large-Scale Social Media Data with Customizable and Scalable Indexing Techniques over NoSQL Databases | 2017 | Columns | Social media | efforts in supporting the data storage and processing requirements | IndexedHBas | Indexed Base | a large number | HBase |
| 9 | An Efficient Index for Massive IOT Data in Cloud Environment | 2017 | Columns | Internet of Things | transactional and indexing extension for H Base | cal- index Secondary index | Secondary index | a large number | HBase |
| 10 | Efficient B-tree Based Indexing for Cloud Data Processing | 2017 | Columns | flexible computing infrastructure | we present a novel, scalable B+-tree based indexing scheme for efficient data processing in the Cloud | Cloud Global index | CG-index | a large number | Cloud |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | Towards Performance Evaluation of HBase based Multidimensional Cloud Index | 2015 | Columns | | data processing and retrieval<br>Is organized as follows. Section 2 introduces the related work of multidimensional cloud indexing Approaches. | multidimensional cloud indexing | MD- HBase techniques | a large number | HBase |
| 12 | A Performance-improved and Storage-efficient Secondary Index for Big Data Processing | 2017 | Columns | smart grid | these indexing techniques are essentially designed for columns, not for the whole query, which slow down the query process | Secondary Index | HIndex and Hive | a large number | HBase |
| 13 | Ingestion, Indexing and Retrieval of High-Velocity Multidimensional Sensor Data on a Single Node | 2017 | Columns | sensors, systems, and automated processes generate (data stream). | Ingestion, Indexing and Retrieval | Tree | R*-tree and the local indices are serialized kd-trees | a large number | HBase |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 14 | A Single-Node Datastore for High-Velocity Multidimensional Sensor Data | 2016 | Columns | Internet of Things | go back to basics and revisit the ingestion, indexing, storage, and retrieval of high-velocity multidimensional sensor data on a single node | Tree | R*-tree | a large number | HBase |

# Chapter three

# Research methodology

### 3.1 Over view:

In this chapter we will explain the methods or methods you used to get to the research problem and the tools you used to get to the problem.

### 3.2 Research Design and Procedure:

The research framework started with literature review, which studies several published papers focused on Nosql database and index techniques. Furthermore, the researcher highlighted the problem of study as well as the technique that used which is hash indexing; in addition to that the researcher explained the designation of the intended technique and prepared the environment and the suitable tools that can be utilized. In addition, the data set and the test are made by the researcher through the research and did the readings, then the researcher undertook the results and then evaluated them.

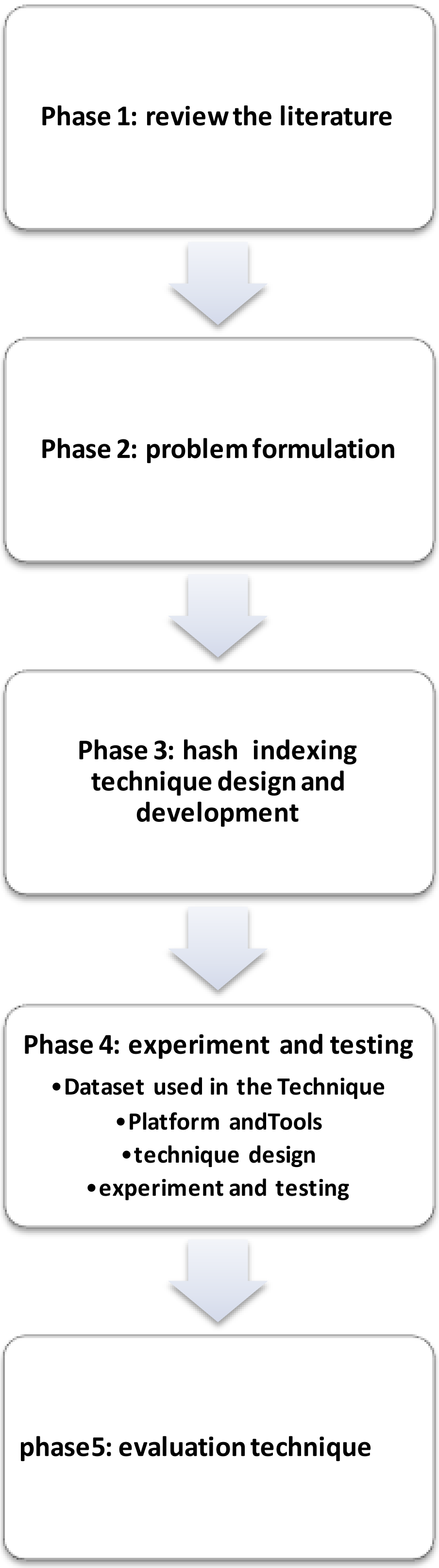

Phase 1: review the literature
Phase 2: problem formulation
Phase 3: hash indexing technique design and development
Phase 4: experiment and testing
•Dataset used in the Technique
•Platform andTools
•technique design
•experiment and testing
phase5: evaluation technique

**Figure 3.1 Research Framework**

### 3.3 problems Formulation:

The development of science and information technology has led to an increase in the quantity of data compiled. the quantities of this data is very large and required to be processed, stored, managed and utilized, After determine problem formulations there are algorithms and techniques give us solution space area. On this area we can determine what exactly techniques design and development the work. Depending on all Steps, we can implement and applying experiment and testing our solution. from here the following steps illustrate this: Literature has been reviewed in the Nosql databases for information, understanding and how to deal with them, as well as review the literature in the indexes in the databases and showed the amount of papers have a second index and hence the techniques used in the second index were compared and the problem of delay in time emerged techniques of indexing the work of research.

### 3.4 Design and Development of hash indexing technique:

After reviewing the literature in the techniques of indexes in traditional databases to know the techniques and choose the appropriate one suitable for databases and understanding and knowledge of the shortcomings of them must be thought of some solutions and hence began the technical Hash index to facilitate the process of index and accelerate the time of retrieval in the process of research.

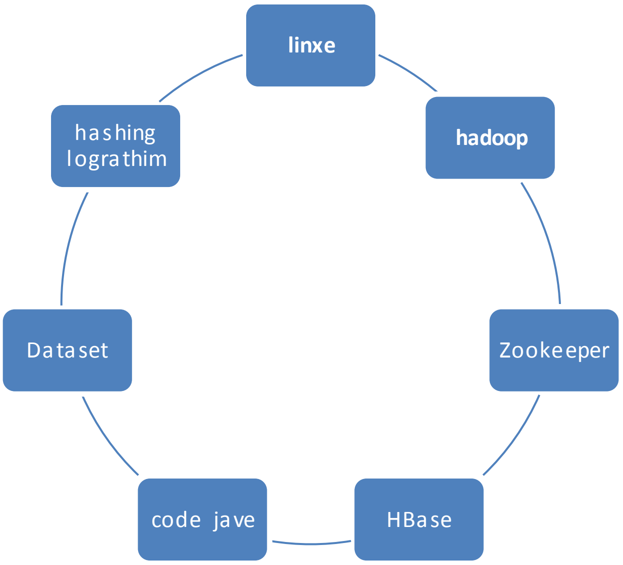


**Figure 3.2 operating system**

## 3.5 Experiment and Testing:

The indexing algorithm implemented and tested on the server is designed with the specifications as explained in Table 3.1, after the environment has been configured and the tools required in the work environment are activated.

**Table 3.1 spec server**

| Server | Processor | RAM | Hard disk | Operating system |
|---|---|---|---|---|
| Hp Compaq Elite Convertible Minitower | Intel (R) Core i5 3.00GHz | 4GB | 320GB | Ubuntu Server 16.04LTS |

### 3.5.1 Platforms and tools:

Since 1970, RDBMS has been the solution for data storage and maintenance related problems. After the advent of big data, companies realized the benefit of processing big data and started opting for solutions like Hadoop.

### 3.5.1.1The Hadoop Distributed File System (HDFS):

Hadoop uses distributed file system suitable for storing large files data, and Map Reduce to process it. Hadoop excels in storing and processing of huge data of various formats such as arbitrary, semi, or even unstructured[43]. Enables the underlying storage for the Hadoop cluster, it divides the data into smaller parts and distributes it across the various servers/nodes, it provides high latency batch processing, It provides only sequential access of data. There are two basic components at the core of Apache Hadoop[44]: the Hadoop Distributed File System (HDFS), and the Map Reduce parallel processing framework[45].

**1. HDFS**: HDFS is the storage component of Hadoop. It's a distributed file system. HDFS is a fault tolerant and self-healing distributed file system designed to turn a cluster of industry standard servers into a massively scalable pool of storage. HDFS is optimized for high throughput and works best when reading and writing large files[46].

**2. Map Reduce:** Map Reduce is a massively scalable, parallel processing framework that works in tandem with HDFS. With Map Reduce and Hadoop, computations are executed at the location of the data.

#### 3.5.1.2 Zookeeper:

Allow centralized infrastructure with various services, providing synchronization across a cluster of servers. Large data analytics applications utilize these services to coordinate parallel processing across big clusters[47]. Apache Zookeeper provides operational services for a Hadoop cluster. Zookeeper provides a distributed configuration service, a synchronization service and a naming registry for distributed systems. Distributed applications use Zookeeper to store and mediate updates to important configuration information[48].

#### 3.5.1.3 HBase:

HBase is a column stored database management system that runs on top of Hadoop Distributed File System (HDFS). It is well suited for sparse data sets, which are common in many big data use cases[44]. HBase applications are written in Java much like a typical Apache Map Reduce application. an HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database[49]. Each table must have an element defined as a

Primary Key, and all access attempts to HBase tables must use this Primary Key. Additionally, HBase supports a rich set of primitive data types including: numeric, binary data and strings[50], and a number of complex types including arrays, maps, enumerations and records.

### 3.5.2Dataset used in the Technique:

The name is data set Social Influence on Shopping is Survey of 2,676 millennial, What social platform has influenced online shopping the most, This data was collected on social survey mobile platform What goodly. Shop have 300,000 millennial and members, and have collected 150,000,000 survey responses from this demographic to date. Data dictionary View, No definitions added for the 1 file and the 6 columns in this dataset [51].

### About this dataset:

**Table 3.2 Description of dataset**

| No | Info | Description |
|---|---|---|
| 1. | Shared with | Everyone |
| 2. | Created | Mar 18, 2017 by @ahalps |
| 3. | Modified | May 18, 2017 · All activity |
| 4. | Version | cdbaa8b1 |
| 5. | Tags | ecommerce, social, social media, online shopping, millennial, marketing, social media marketing, snap chat, instagram, twitter, facebook |
| 6. | License | CC-BY-SA |

### 3.5.3 Technique design:

When configuring the working environment and running the tools, a Hash function is applied and a Hash table is created to store Hash values or Hash symbols. The process of fragmentation of data is properly stored and retrieved to determine the speed of execution of the query

### 3.5.4 Experiment and Testing:

#### 3.5.4.1Experiment

Run Hash function and make some modifications to it in the relational database. It relies on fragmentation, reducing the number of keys in the index of the Hash table, and converting the index data type to numeric to facilitate the search. We used Hash function mainly in Hash table to quickly locate data and map HASH values Possible to reduce the number of collisions in the index.

#### 3.5.4.2Testing:

Perform the test after making sure that the Hash function performs a retrieval of data from the dataset by searching, recording the data retrieval time and interpreting the results.

## 3.6Evaluation technique:

Comparison of the results of the original data retrieved from the non-indexed data bases with the results retrieved from the Hash table after the implementation of Hash and some time measurement after reducing the number of collisions. To make sure that Hash works effectively, it is necessary to make calculations easier, fewer seals and a uniform distribution of data.

# Chapter four

# Index technique based on hash algorithm implementation and results

## 4.1 Over view:

This chapter consists of two parts. The first section examines the technique of Hashing, Hash function, Hash table and its features, and the second section, how to design the Hash technique, how to conduct the experiment, Test experiment, discuss the results and evaluate them.

## 4.2 Hash as concept:

The Hashing is considered as the most important and fastest data structures ever, and many applications use this structure in compilers. The hash structure quickly accesses any data, regardless of the size of this data as well as data entry very quickly. In addition to the speed feature, it is easy to apply as applying it through a regular matrix or a coefficient and convert the large keys to small keys. In general, it is possible to convert a set of keys to specific locations in the index in the array. In the simplest equation, there is no conversion process and the key is the same as the direct index in the matrix. But there are many other cases where there is no key from the basis and therefore the process of conversion from any value to the index sites will be done using the Hash function and in case index type is the integer number. In hashing there is a hash function that maps keys to some values. But this hashing function may lead to collision that is two or more keys are mapped to same value, Hashing is implemented in two steps:

- ➢ An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.
- ➢ The element is stored in the hash table where it can be quickly retrieved using hashed key.

### 4.2.1 Features of Hashing:

Hash functions are used to meet some of the properties listed below[52]:

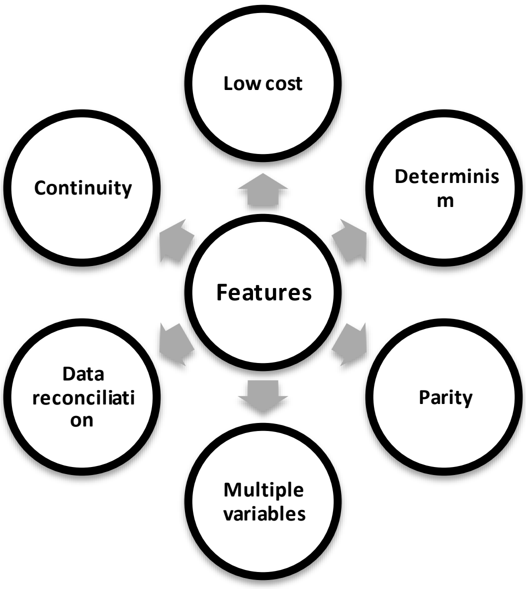


**Figure 4.1Features of Hashing**[52]

- **Low cost:** The costs of creating a Hash function must be low enough to find a useful solution based on the Hash function compared to alternative methods.
- **Determinism:** An hash function must be deterministic - that is, each value of the inputs must produce the same value as the hash. The function must be Hash in the arithmetic sense of the term. This condition does not apply to hash functions that depend on external variables.
- **Parity:** The Hash function must specify the expected inputs as evenly as possible through the output set. This means that each Hash value must be produced with the same probability.
- **Multiple variables:** In many applications, the hash values may vary at each run of a program, or may change during the same operation (when an hash table needs to be expanded). In this case, one needs a hash function with two variable operators - the input data z and the n number of allowed hash values.
- **Data reconciliation:** Some data entered in some applications may contain features that are not related to comparison purposes.

- **Continuity:** The hash function used to find similar data must be as continuous a function as possible; two inputs that differ slightly at approximately equal hash value must be specified.

### 4. 2. 2 The uses of Hash:

In practice are used in several ways[53]:

1. **Test integration or no change in data:** through the Hash to any data you will get the exit output. In case you take the Hash for the same data, we must get the same Hash output. If Hash changes, this means that the data has changed.
2. **Search:** For example, I want the same files in a disk, the easiest way is to calculate each file. And the knowledge of the matching hacks and so on.
3. **The data is stored in a non-retrievable manner:** it is known that the two keys must be stored in the base in the form of a hash. The reason is that if there is a penetration of the base, all the passwords will be easily retrieved, so it was like a trap for the hacker.

## 4. 3 Hash algorithm:

Hashing Algorithm is using this study for the storage of the unstructured data into the HBase database. Thus the consisting hashing algorithm is very useful to store the large amount of unstructured data into the HBase database. The steps of Hash[54]:

- Tables are used to store data or records, and retrieve them quickly.
- Records are stored in groups using the hash keys.
- Hash keys are calculated by applying the hash algorithm to a selected value within the record. This value must be a shared value for all records.
- Each group can contain organized records in a particular order.

**Algorithm 1: Hash index algorithm [55].**

```
Input: A ← key from dataset table
Output K: Hash index key
Function Hash Index (A)
{
 Keysarray [ DatasetKey];
 DatasetKey← ASCI (A);
Hash Index=(Sum(DatasetKey) mod Keysarray_Size );
Hash Index -> K
}
```

### 4. 4 Hashing function:

A hash function is any function that can be used to map data of arbitrary size to data of a fixed size[56] . The values returned by a hash function are called hash. Hash functions are often used in combination with a hash table, a common data structure used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file.

Hash function assigned key values to elements in a hash table using a Hash function is helps calculate the best index an item should going. Index must be small enough for the arrays size[57]. Don’t over write other data in the Hash Table. Is job to store values in an array with a limited size. It does it in a way that the array doesn’t need to be searched through to find it[58].

### 4. 5 Indexing technique based on hash:

Hashing used to uniquely identify a specific object from a group of similar objects. Assume that you have an object and you want to assign a key to it to make searching easy. To store the key/value pair, you can use a simple array like a data structure where keys (integers) can be used directly as an index to store values. However, in cases where the keys are large and cannot be used directly as an index, you should use hashing. In hashing, large keys are converted into small keys by using hash functions. The values are then stored in a data structure called hash table. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key). By using that key you can access the element in O (1) time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

### 4. 6 Framework for Environment indexing technique:

One of the challenges faced this research find a tools that are allow the user to deal with Nosql databases. Because the Nosql is new technology as well as there are no standard till now, hence each Nosql DBMS follow different method to deal with the database. Consequently, to find driver or tool to connect the application to DBMS is very difficult, because the programming languages don't provide a library for each DBMS's. additionally, there are not any test bed or application that allow the research to test their proposed indexing algorithm.

The aforementioned issues directed the researcher to develop a framework that will help other researchers to test their indexing algorithms in an easy way.

The framework is composed from two parts as explained in Figure 4.2. dataset part and the algorithm part.

1. The **data set:** this part allows the researcher to insert their dataset into database table in HBase, which the dataset is read from the file and then inserted into database.
2. The **indexing** process: framework provides a function that allows the researcher to write their algorithm code inside it. Additionally the framework connects the proposed algorithm with user's dataset to index the data.
3. Additionally, the framework offer a simple interface to perform the user's queries and display the result of the query as well as calculate the response time before show it to the user.

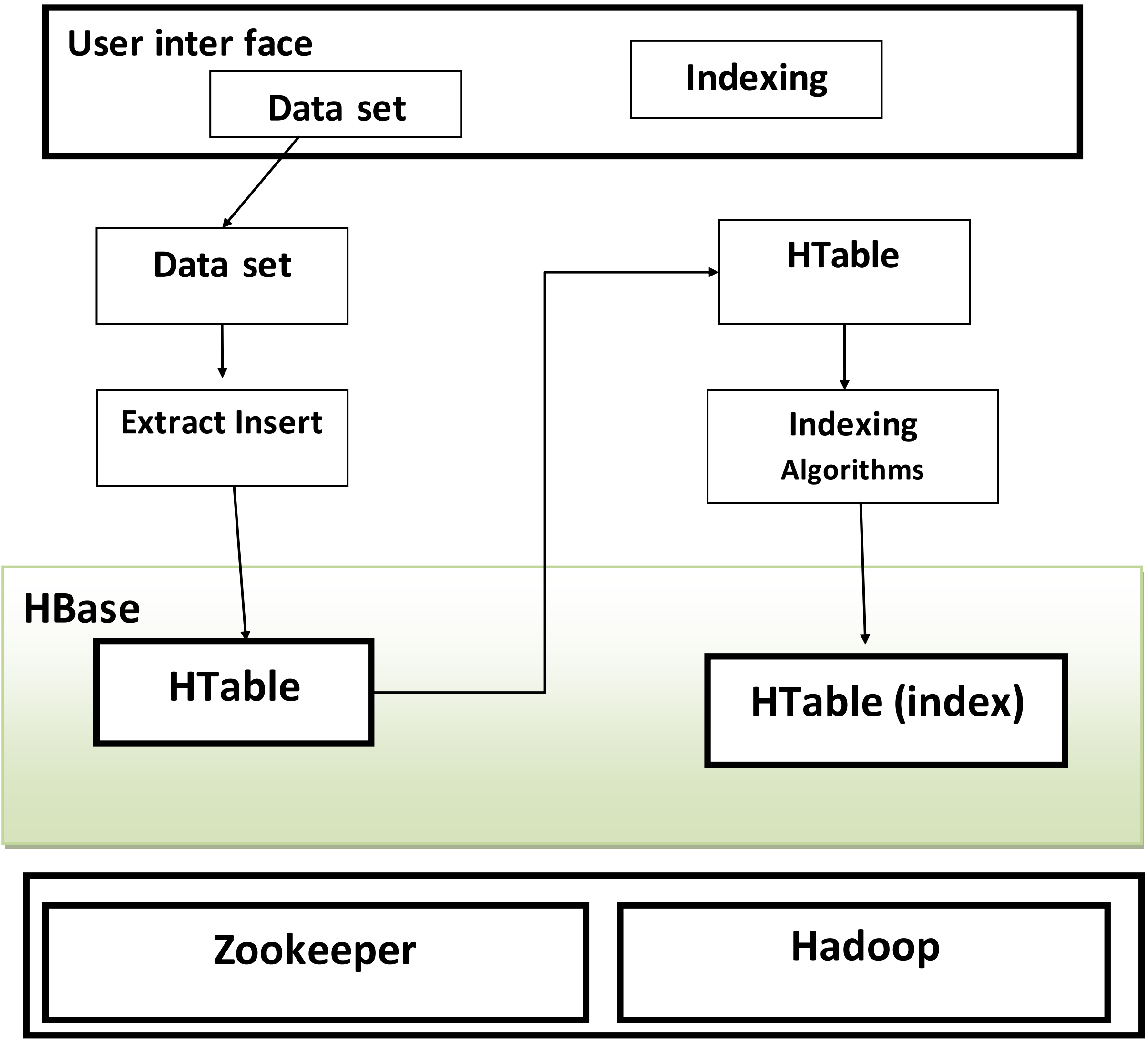


Figure 4.2 Frameworks for Environment Indexing technique

**Experiment and discussion:**
**Environment setup from HBase:**

The workspace is equipped with the Hadoop, an open source library, the Zookeeper utility to connect Hadoop to the HBase database, the Java code the data in the HBase file is sat with what is known as the data table. Then, take the reading from the table. Then measure the recovery time by the search process. In hash technology, a hash table is fed through arithmetic and Hash function, measuring time by calculating keys, measuring recovery time, and observing the time difference after the recovery process from the hash table.

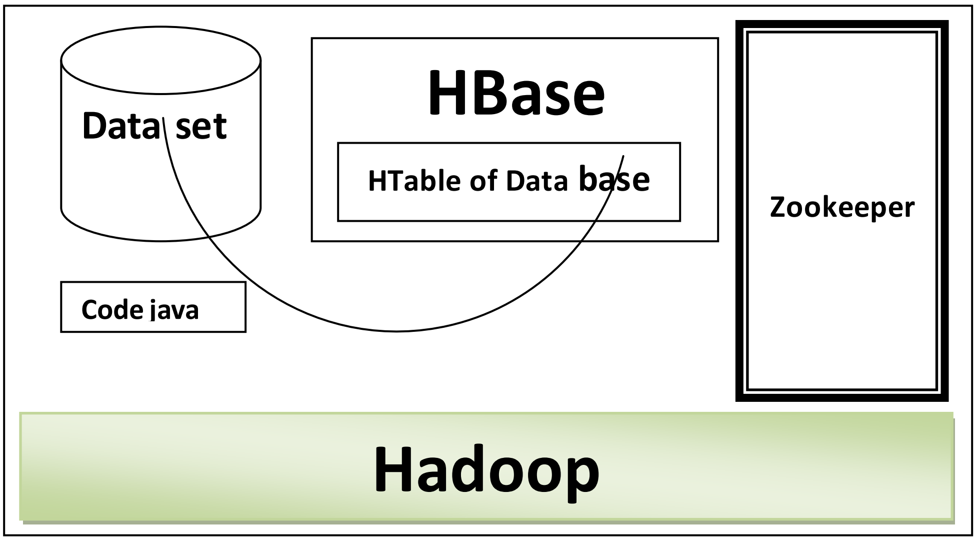


Figure 4.3Architecture in Environment

## 4. 7 Description of the indexing process:

The new index is created in a matrix using the Hash algorithm by calculation of the Hash function and stored in the Hash table in the form of pairs. The algorithm is implemented in three stages as described by Down:

### Created Hash Table:

A Hash table is created which stores the values, which are the old index and the index of the Hash, Hash Table Is a data structure that is used to store key pairs and uses a hash function to calculate indexing in a matrix in which an element or search is managed using a good hash function[59], offers fast insertion and search, they are limited in size because, they are based on arrays, can be resized, but it should be avoided. They are hard to order[60]

### Generating new index using Hash Function:

Use in Hash Algorithm: Calculation applied to a key to transform it into an address, for numeric keys, divide the key by the number of available addresses, n, and take the remainder for alphanumeric keys, divide the sum of ASCII codes in a key by the number of available a addresses, n and take the remainder, Folding method divides key into equal parts then adds the parts together.

### Store the new index with old index in pairs:

The new index is stored after attaching the Hash with the old index in pairs called the values inside the Hash table.

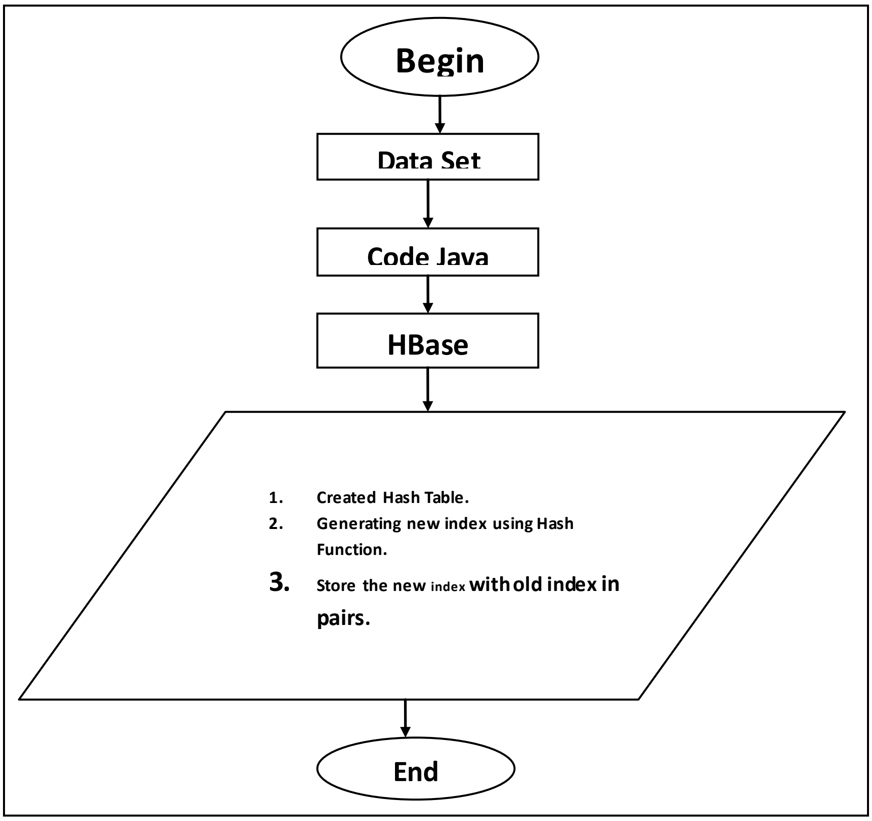

Begin
Data Set
Code Java
HBase
1. Created Hash Table.
2. Generating new index using Hash Function.
3. Store the new index with old index in pairs.
End

Figure 4.4 Flow chart Store Data in HBase using Hashing

## 4. 8 Query for retrieval process on hash technique:

The process of retrieving the values after the Calculation index user keys using Hash Function operation, Provides the Exact index for the Information, Goes directly to the location in the array and Send Info back to the user. Then Get old index from Hash Table using new index, and Get row record from data set Table using old index. As shown in figure below:

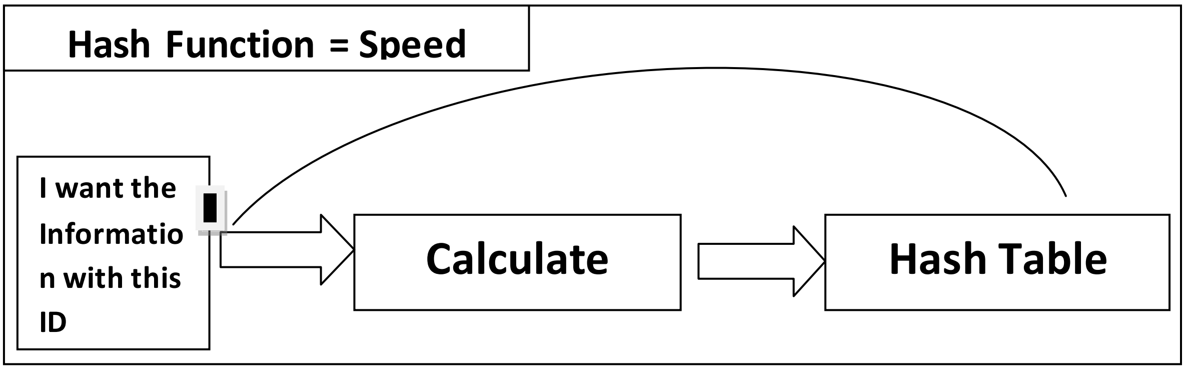


Figure 4.5 How to retrieval in hashing

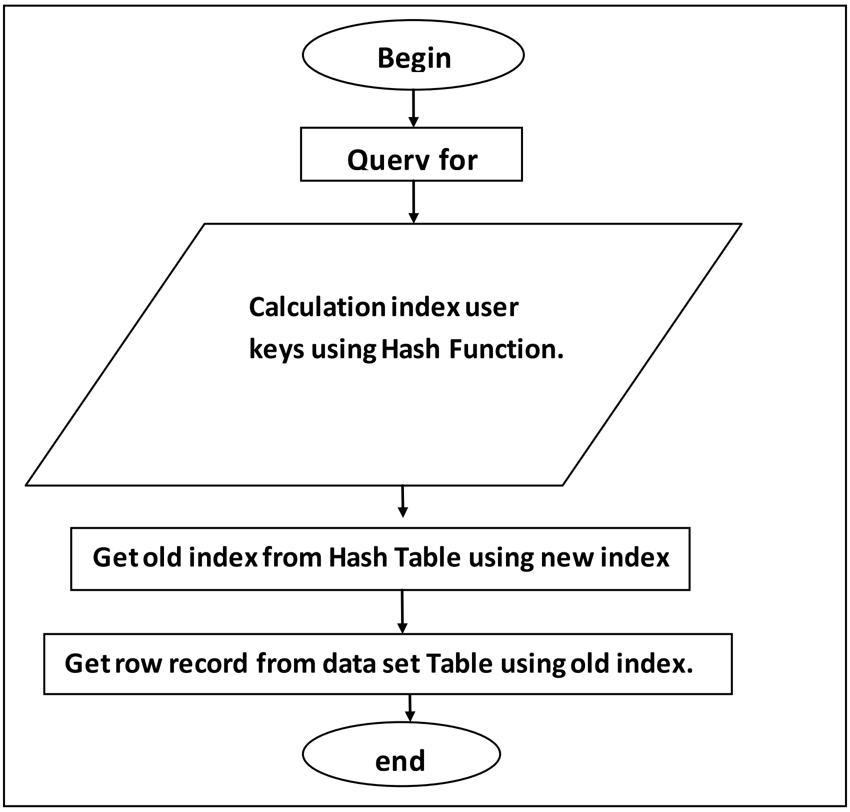

Begin
Querv for
Calculation index user keys using Hash Function.
Get old index from Hash Table using new index
Get row record from data set Table using old index.
end

Figure 4.6 Flow chart retrieval processes

## 4. 9 Experiment and Results:

In this section, the experiment and Results is conducted to evaluate performance of index technique on Social Influence Data set. This Data set contains Social Influence Form a lot of HBase. Here, retrieval main Data which segmented to 10 main body regions.

## 1. Experiment one:

The experiment start by retrieving large data Different sizes of Data Set from HBase were retrieved and read back time was retrieved. The problem of latency occurred in time because the retrieval process was done through the search process. The Table 4.1 demonstrates the Result:

**Table 4.1 Response time from HBase**

| **No of record per millions** | Indexed algorithm **response time in seconds** |
|---|---|
| 1 | 16 |
| 2 | 26 |
| 3 | 31 |
| 4 | 39 |
| 5 | 50 |
| 6 | 54 |
| 7 | 64 |
| 8 | 72 |
| 9 | 79 |
| 10 | 83 |

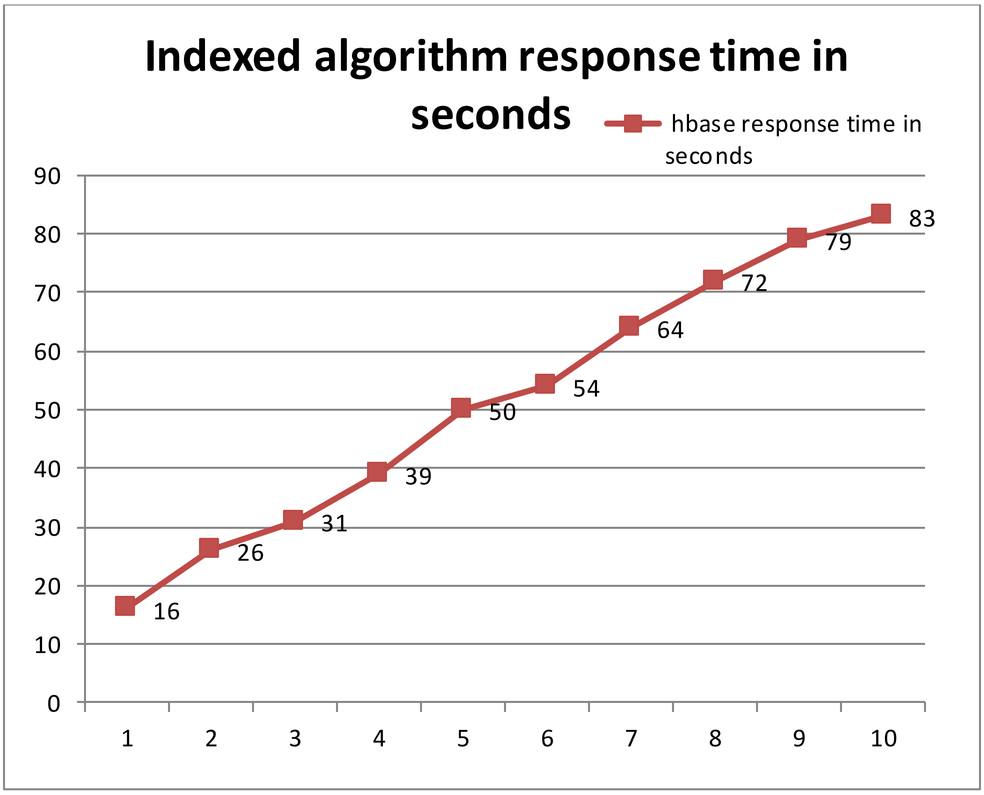


**Figure 4.7response time from HBase**

### Result one:

Larage data retrieved Indexed algorithm is Delay in response time in seconds time. As indicated in Appendix No(1, 2).

### 2. Experiment two:

After the download of Hash tables and the Hash function, the data was retrieved through a calculation using the Hash function. The time was measured and the recovery time was reduced from the comparison tables of Data From Indexed algorithm but the decrease in time is very small because the number of keys is large in the index Keys. The next table demonstrates the Result:

Table 4.2Response time from hash function

| No of record per millions | Hash function response time in seconds |
|---|---|
| 1 | 14 |
| 2 | 24 |
| 3 | 28 |
| 4 | 37 |
| 5 | 46 |
| 6 | 52 |
| 7 | 61 |
| 8 | 69 |
| 9 | 71 |
| 10 | 75 |

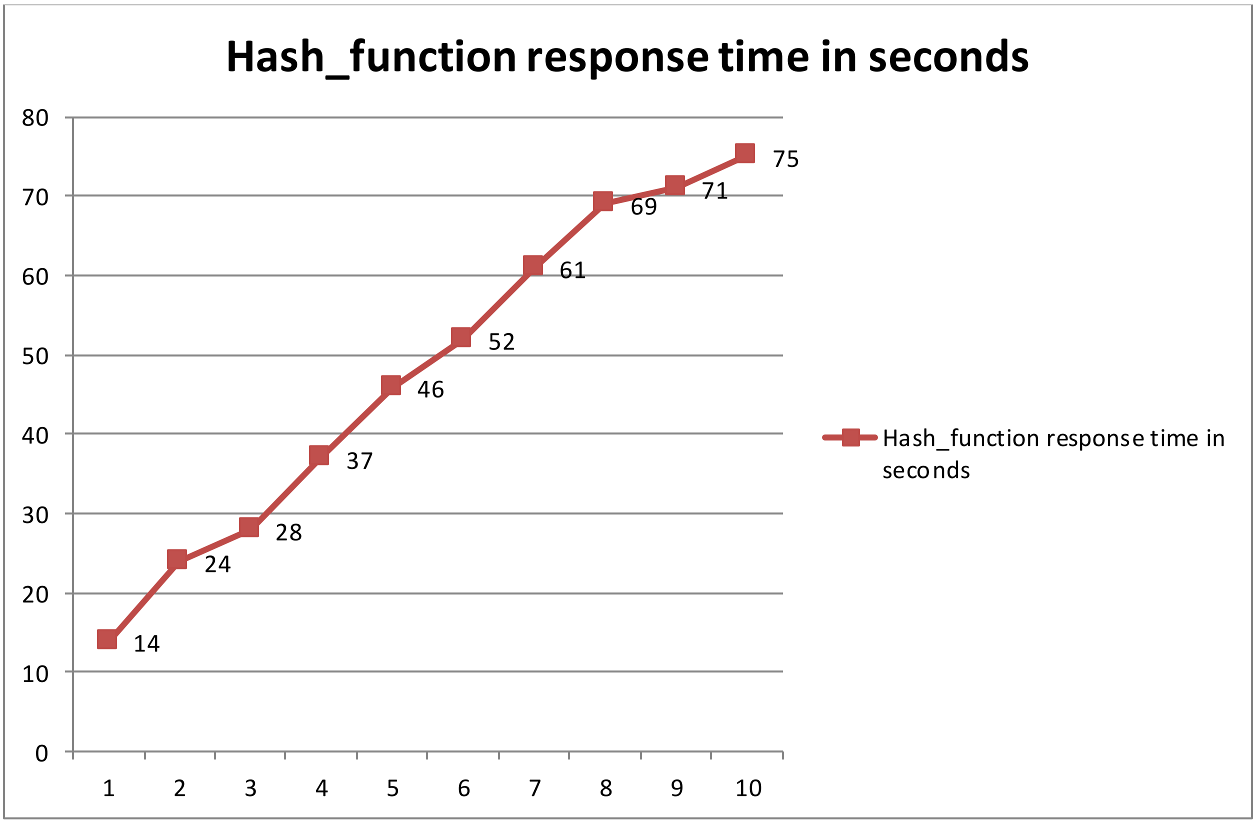


Figure 4.8 response time from Hash function

**Result two:**

When Hash Algorithm is used in retrieval the reduced is response time. As indicated in Appendix No(3,4)

## 3 Comparison between retrieval times of the data obtained from Indexed in Htable and Hash function in Hash table :

The retrieval time of the data obtained from indexed in Htable the search process and the retrieval time of the data from the Hash table were compared by a calculation of the Hash function. The next table demonstrates the Result:

Table 4.3Response time from HBase and hash function

| No of record per millions | HBase response time in seconds | Hash function response time in seconds |
|---|---|---|
| 1 | 16 | 14 |
| 2 | 26 | 24 |
| 3 | 31 | 28 |
| 4 | 39 | 37 |
| 5 | 50 | 46 |
| 6 | 54 | 52 |
| 7 | 64 | 61 |
| 8 | 72 | 69 |
| 9 | 79 | 71 |
| 10 | 83 | 75 |

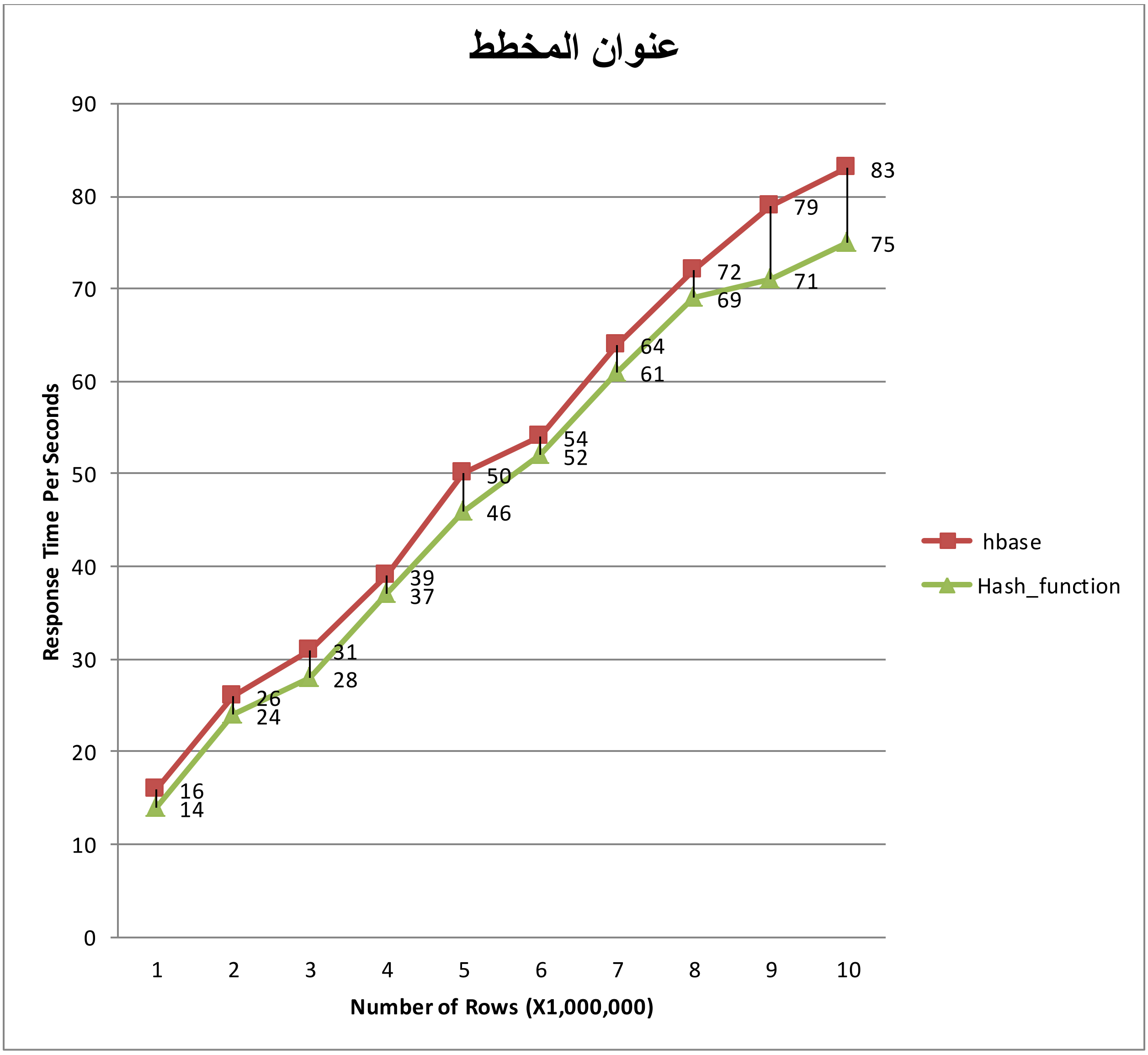


**Figure 4.9 response time from HBase and hash function**

## Result three:

Search in indexed algorithm retrieval delay the response time where ape calculation Hash Algorithm used Hash Function retrieval reduces the response time. As indicated in Appendix No (5,6).

### 4. 10 Discussion of Hash indexed technique:

After the constructed of Hash tables using the Hash function, the data was retrieved through a calculation using the Hash function. The time was measured and the recovery time was reduced from the comparison tables of Data From HBase but the decrease in time is very small because the number of keys is large in the index Keys.

### In experiment 1:

After the initialization of the environment and the work of retrieving the data from the database in different sizes and the size of the first reading 100 Record and took ten volumes in a row and recording the recovery time through the drawing was noted that the larger the size of the data increased the recovery time, which confirms the delay in time was noted for lack of index in NoSQL there is a primly key In the database, a process is searched through row key in the database, which leads to delay in the search process because of the large data. Here an algorithm is designed to assist the query in executing the process.

### In experiment 2:

The algorithm of Hash was selected and executed on Nosql. The calculation of the Hash table was performed by the Hash function, which resulted in the acceleration of the search process and the readings of the same data volumes were taken from the data set.

### In experiment 3:

The data read from the database was compared with the reading from the Hash table. A slight difference was observed in the collision of the keys and the collision occurs when two elements in the index are in the Hash table, the large number of data keys and the detection of similar rows in a large file.

## 4. 11 Evaluation:

In the database table, the retrieval process takes place after the search process through the row key. In order to increase the number of large keys, there is a delay in the retrieval time. In Hash tables, the process of retrieval is performed through the calculation. In a short time, the Hash table is performed through a Hash function. The Hash function is evaluated by a lower collision, a simple calculation and a uniform distribution. In the string, the number of large keys decreases to smaller keys and fewer collisions speed up query execution.

# Chapter five:

# Conclusion

## 5. 1 Over view:

This chapter the conclusions from this thesis. In section 5.1, we provide summary of the thesis. Future work is proposed in section 5.2.

## 5. 2 Conclusions:

In this thesis, we have studied, analyzed, designed and implementation hash technique for indexing large data As we have seen from the results obtained by many works in the literature, large database has a properties associated with it like volume, velocity, variability and value, but when facing increasing size, it will face delay problems. In addition, unstructured format of data which accused huge hidden values from large datasets with various types and rapid generation and deals with the speed of data from different sources are main problem of large data. To address this issue, we proposed Index technique based on hash algorithm. At first, we studied hash indexing technique design and development it We found in Hash technology that Hash tables are fed through a calculation using the Hash function, as well as retrieving a calculation which makes it easier) Reduce keyboard collisions by increasing storage space in the index table, detecting similar rows, and detect duplicate records in a large file, Contribute to the detection of existing problems in NoSQL database and find appropriate solutions, Work development.

## 5. 3 Future work:

Following the investigation described in this thesis, the main lines of the research re-mains open and number of projects could be taken up:

- Develop and algorithm to handle collision in the hash table such as chain.
- Implement hash indexing technique in the other Nosql type such as key-value store.

## 5. 4 Recommendations:

The Nosql is new technology as well as there are no standard till now, hence each Nosql DBMS follow different method to deal with the database.

# References:

[1] B. Acharya, A. K. Jena, J. M. Chatterjee, R. Kumar, and D.-N. Le, "NoSQL Database Classification: New Era of Databases for Big Data," *International Journal of Knowledge-Based Organizations (IJKBO),* vol. 9, pp. 50-65, 2019.

[2] L. Perkins, E. Redmond, and J. Wilson, *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*: Pragmatic Bookshelf, 2018.

[3] A. Nayak, A. Poriya, and D. Poojary, "Type of NOSQL databases and its comparison with relational databases," *International Journal of Applied Information Systems,* vol. 5, pp. 16-19, 2013.

[4] R. R. Parmar and S. Roy, "MongoDB as an Efficient Graph Database: An Application of Document Oriented NOSQL Database," *Data Intensive Computing Applications for Big Data,* vol. 29, p. 331, 2018.

[5] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," *ACM sIGKDD Explorations Newsletter,* vol. 14, pp. 1-5, 2013.

[6] J. Liu, J. Li, W. Li, and J. Wu, "Rethinking big data: A review on the data quality and usage issues," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 115, pp. 134-142, 2016.

[7] K. A. I. Hammad, M. A. I. Fakharaldien, J. M. Zain, and M. A. Majid, "Big Data Analysis and Storage."

[8] S. Schmid, E. Galicz, and W. Reinhardt, "Performance investigation of selected SQL and NoSQL databases," *AGILE 2015–Lisbon,* pp. 9-12, 2015.

[9] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation," *Journal of Big Data,* vol. 2, p. 18, 2015.

[10] V. Chang, Y.-H. Kuo, and M. Ramachandran, "Cloud computing adoption framework: A security framework for business clouds," *Future Generation Computer Systems,* vol. 57, pp. 24-41, 2016.

[11] W. Cho and E. Choi, "A GPS trajectory map-matching mechanism with DTG big data on the HBase system," in *Proceedings of the 2015 International Conference on Big Data Applications and Services*, 2015, pp. 22-29.

[12] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, 2011, pp. 363-366.

[13] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *Cloud and Service Computing (CSC), 2011 International Conference on*, 2011, pp. 336-341.

[14] K. Fan, "An overview of NoSQL database," *Programmer,* vol. 6, pp. 76-78, 2010.

[15] B. Acharya, M. Pandey, and S. S. Rautaray, "SURVEY ON NoSQL DATABASE CLASSIFFICATION: NEW ERA OF DATABASES FOR BIG DATA."

[16] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191,* 2013.

[17] B. Shen, Y.-C. Liao, D. Liu, and H.-C. Chao, "A Method of HBase Multi-Conditional Query for Ubiquitous Sensing Applications," *Sensors,* vol. 18, p. 3064, 2018.

[18] H. Zhuang, K. Lu, C. Li, M. Sun, H. Chen, and X. Zhou, "Design of a more scalable database system," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 1213-1216.

[19] G. Cox, E. Armengaud, C. Augier, A. Benoit, L. Bergé, T. Bergmann*, et al.*, "A multi-tiered data structure and process management system based on ROOT and CouchDB," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment,* vol. 684, pp. 63-72, 2012.

[20] D. D. Tambunan, "PERBANDINGAN ANALISIS APLIKASI DATABASE NoSQL REDIS DAN SQL MySQL," Universitas Widyatama, 2016.

[21] A. K. Zaki, "NoSQL databases: new millennium database for big data, big users, cloud computing and its security challenges," *International Journal of Research in Engineering and Technology (IJRET),* vol. 3, pp. 403-409, 2014.

[22] P. Murugan and A. Sentraya, "A Study of NoSQL and NewSQL databases for data aggregation on Big Data," ed, 2013.

[23] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 512-521.

[24] Y. Liu, F. Cao, M. Mortazavi, M. Chen, N. Yan, C. Ku*, et al.*, "DCODE: A distributed column-oriented database engine for big data analytics," in *Information and Communication Technology*, ed: Springer, 2015, pp. 289-299.

[25] A. Nowosielski, P. A. Kowalski, and P. Kulczycki, "The column-oriented database partitioning optimization based on the natural computing algorithms," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2015, pp. 1035-1041.

[26] A. Nowosielski, P. A. Kowalski, and P. Kulczycki, "The column-oriented database partitioning optimization based on the natural computing algorithms," in *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*, 2015, pp. 1035-1041.

[27] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, p. 469.

[28] R. Dharavath, A. K. Jain, C. Kumar, and V. Kumar, "Accuracy of atomic transaction scenario for heterogeneous distributed column-oriented databases," in *Intelligent Computing, Networking, and Informatics*, ed: Springer, 2014, pp. 491-501.

[29] C. Cao, W. Wang, Y. Zhang, and X. Ma, "Leveraging Column Family to Improve Multidimensional Query Performance in HBase," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, 2017, pp. 106-113.

[30] S. Gugnani, X. Lu, H. Qi, L. Zha, and D. K. D. Panda, "Characterizing and accelerating indexing techniques on distributed ordered tables," in *Big Data (Big Data), 2017 IEEE International Conference on*, 2017, pp. 173-182.

[31] Y. Zou, J. Liu, S. Wang, L. Zha, and Z. Xu, "CCIndex: A complemental clustering index on distributed ordered tables for multi-dimensional range queries," in *IFIP International Conference on Network and Parallel Computing*, 2010, pp. 247-261.

[32] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 237-248.

[33] C. Feng, X. Yang, F. Liang, X.-H. Sun, and Z. Xu, "Lcindex: A local and clustering index on distributed ordered tables for flexible multi-dimensional range queries," in *Parallel Processing (ICPP), 2015 44th International Conference on*, 2015, pp. 719-728.

[34] C. Wang, Y. Zhu, Y. Ma, M. Qiu, B. Liu, J. Hou*, et al.*, "A query-oriented adaptive indexing technique for smart grid big data analytics," *Journal of Signal Processing Systems,* vol. 90, pp. 1091-1103, 2018.

[35] Y. Guo, S. Li, H. Zhang, and W. Zhong, "A HBase Secondary Index Method Based on Isomorphic Column Family."

[36] H. Qi, X. Chang, X. Liu, and L. Zha, "The consistency analysis of secondary index on distributed ordered tables," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, 2017, pp. 1058-1067.

[37] X. Gao and J. Qiu, "Supporting queries and analyses of large-scale social media data with customizable and scalable indexing techniques over NoSQL databases," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, 2014, pp. 587-590.

[38] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang*, et al.*, "An efficient index for massive IOT data in cloud environment," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 2129-2133.

[39] S. Wu, D. Jiang, B. C. Ooi, and K.-L. Wu, "Efficient B-tree based indexing for cloud data processing," *Proceedings of the VLDB Endowment,* vol. 3, pp. 1207-1218, 2010.

[40] S. Liang and Y. Yang, "Towards performance evaluation of HBase based multidimensional cloud index," in *Computer Science and Network Technology (ICCSNT), 2015 4th International Conference on*, 2015, pp. 629-632.

[41] J. A. Colmenares, R. Dorrigiv, and D. G. Waddington, "Ingestion, Indexing and Retrieval of High-Velocity Multidimensional Sensor Data on a Single Node," *arXiv preprint arXiv:1707.00825,* 2017.

[42] J. A. Colmenares, R. Dorrigiv, and D. G. Waddington, "A single-node datastore for high-velocity multidimensional sensor data," in *Big Data (Big Data), 2017 IEEE International Conference on*, 2017, pp. 445-452.

[43] A. Katal, M. Wazid, and R. Goudar, "Big data: issues, challenges, tools and good practices," in *2013 Sixth international conference on contemporary computing (IC3)*, 2013, pp. 404-409.

[44] M. N. Vora, "Hadoop-HBase for large-scale data," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, 2011, pp. 601-605.

[45] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," in *BMC bioinformatics*, 2010, p. S1.

[46] A. Jangra, V. Bhatia, U. Lakhinaza, and N. Singh, "An efficient storage framework design for cloud computing: Deploying compression on de-duplicated No-SQL DB using HDFS," in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, 2015, pp. 55-60.

[47] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," in *USENIX annual technical conference*, 2010.

[48] F. P. Junqueira and B. C. Reed, "The life and times of a zookeeper," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, 2009, pp. 46-46.

[49] N. Das, S. Paul, B. B. Sarkar, and S. Chakrabarti, "NoSQL Overview and Performance Testing of HBase Over Multiple Nodes with MySQL," in *Emerging Technologies in Data Mining and Information Security*, ed: Springer, 2019, pp. 269-279.

[50] A.-V. Vo, N. Konda, N. Chauhan, H. Aljumaily, and D. F. Laefer, "Lessons learned with laser scanning point cloud management in Hadoop HBase," in *Workshop of the European Group for Intelligent Computing in Engineering*, 2018, pp. 231-253.

[51] data.world. (May 18, 2017). *Social Influence on Shopping*. Available: https://data.world/ahalps/social-influence-on-shopping

[52] J. Song, L. Gao, L. Liu, X. Zhu, and N. Sebe, "Quantization-based hashing: a general framework for scalable image and video retrieval," *Pattern Recognition,* vol. 75, pp. 175-187, 2018.

[53] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," 2070-1721, 2018.

[54] H. Yoon, J. Yang, S. F. Kristjansson, S. E. Sigurdarson, Y. Vigfusson, and A. Gavrilovska, "Mutant: Balancing storage cost and latency in LSM-tree data stores," *ACM SoCC,* pp. 162-173, 2018.

[55] A. Kunthavai, S. Vasantharathna, and S. Thirumurugan, "Pairwise Sequence Alignment using Bio-Database Compression by Improved Fine Tuned Enhanced Suffix Array," *International Arab Journal of Information Technology (IAJIT),* vol. 12, 2015.

[56] Y. Li and G. Ge, "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security," *Multimedia Tools and Applications,* pp. 1-22, 2019.

[57] N. B. Greenfield, "System and method for characterizing data through a probabilistic data structure," ed: Google Patents, 2018.

[58] B. J. Bulkowski and V. Srinivasan, "Data distribution across nodes of a distributed database base system," ed: Google Patents, 2018.

[59] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 489-504.

[60] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen*, et al.*, "Exploiting a novel algorithm and GPUs to break the ten quadrillion pairwise comparisons barrier for time series motifs and joins," *Knowledge and Information Systems,* vol. 54, pp. 203-236, 2018.

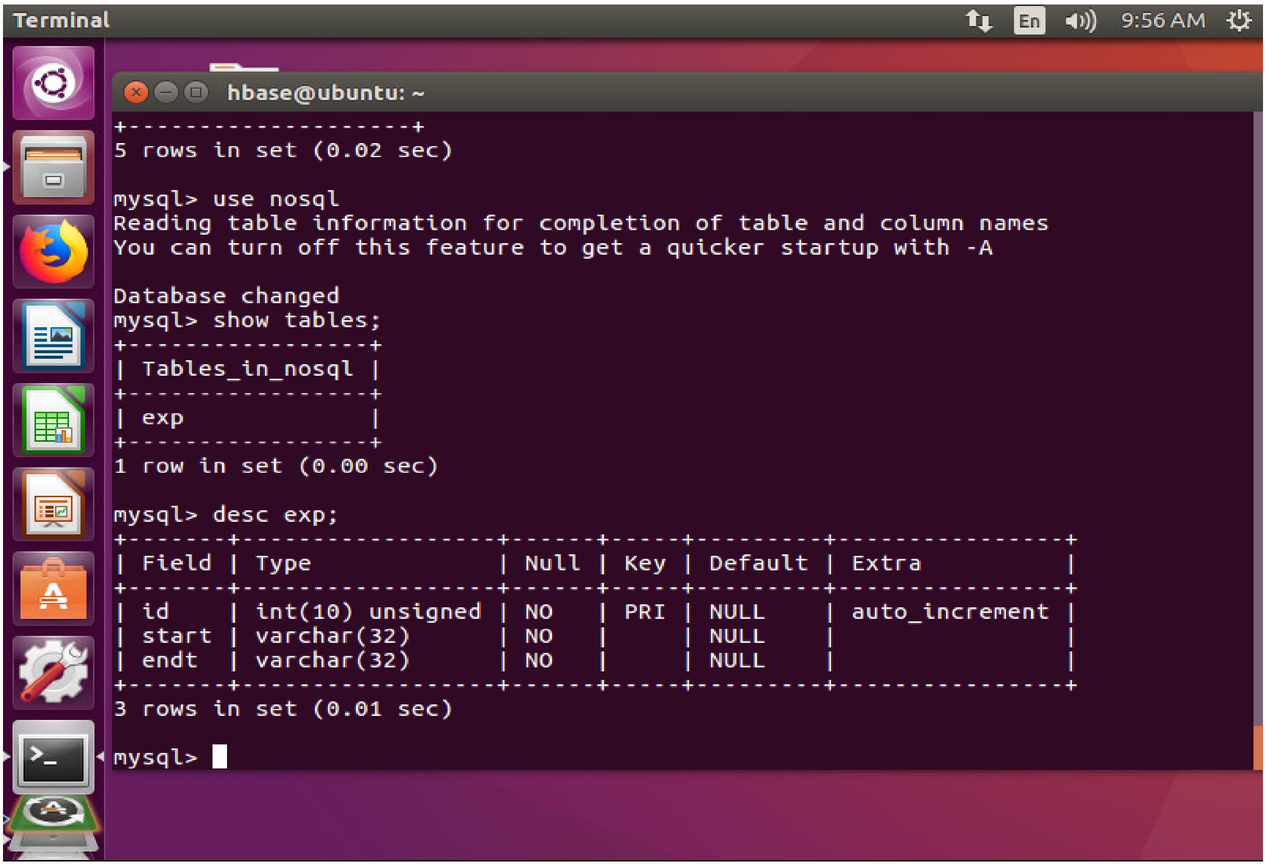
## Appendices

### Histogram equalization java code:

```
Package hashing1;
import java.io.IOException;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
//import
org.apache.hadoop.hbase.protobuf.generated.ClientProt
os.Scan;
import org.apache.hadoop.hbase.util.Bytes;

import com.google.common.hash.HashFunction;

import java.sql.*;

public class RetriveData_old {

    public static void main(String[] args) throws
IOException, Exception {

        //

        Connection conn = null;
        String url = "jdbc:mysql://localhost:3306/";
        String dbName = "nosql";
        String driver = "com.mysql.jdbc.Driver";
        String userName = "root";
        String password = "Qwerty2012";
        Statement st = null;
        Class.forName(driver).newInstance();
        conn = DriverManager.getConnection(url +
dbName, userName, password);
        System.out.println("connected!.....");
        // measuring the start time of the file
download
        long lStartTime = System.nanoTime();
        String query = " insert into exp
(start,endt)" + " values (?, ?)";

        PreparedStatement preparedStmt =
conn.prepareStatement(query);
        preparedStmt.setLong(1, lStartTime);

        // Instantiating Configuration class
        Configuration comfit =
HBaseConfiguration.create ();

        // Instantiating Table class
        Table table = new Table (comfit, "dataset5");

        // Instantiating Get class
        Get g = new Get(Bytes.toBytes("Question"));

        // Reading the data
        Result result = table.get(g);

        // Reading values from Result class object
        byte[] value =
result.getValue(Bytes.toBytes("Question"),
Bytes.toBytes(""));
        byte[] value1 =
result.getValue(Bytes.toBytes("Segment Type"),
Bytes.toBytes(""));
        byte[] value2 =
result.getValue(Bytes.toBytes("Segment Description"),
Bytes.toBytes(""));
        byte[] value3 =
result.getValue(Bytes.toBytes("Answer"),
Bytes.toBytes(""));
        byte[] value4 =
result.getValue(Bytes.toBytes("Count"),
Bytes.toBytes(""));
```

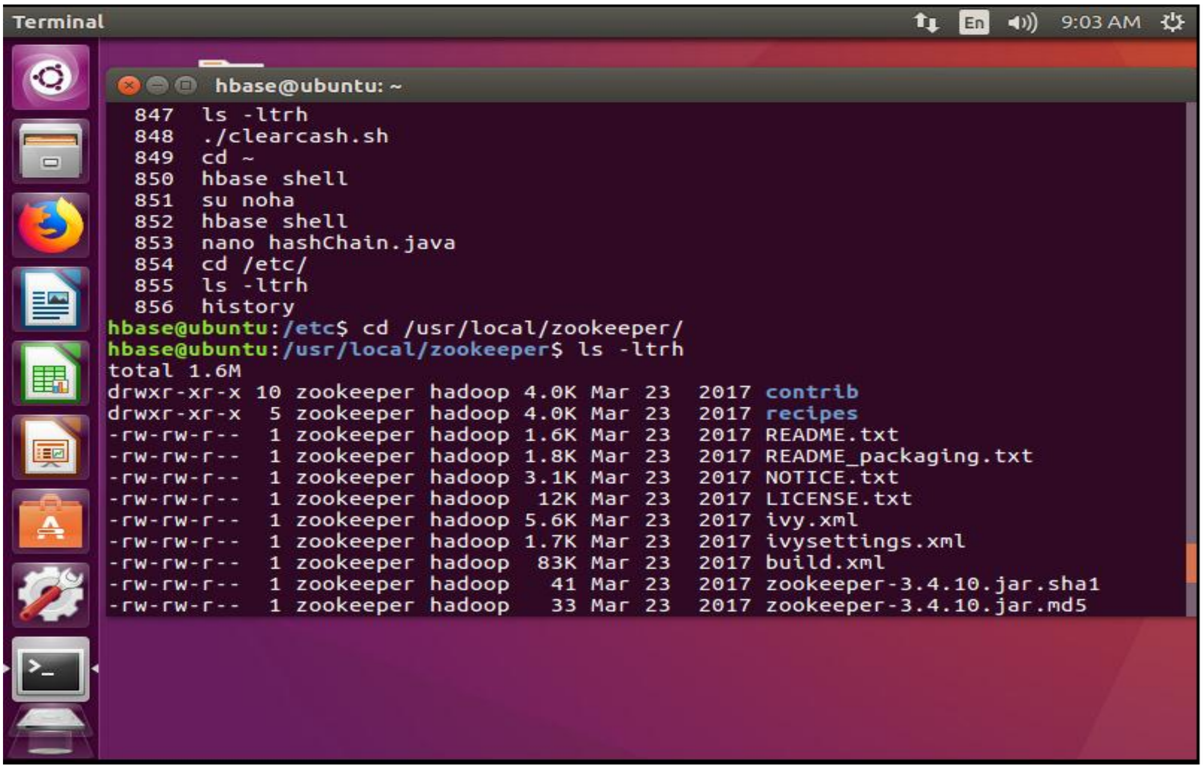
```
        byte[] value5 =
result.getValue(Bytes.toBytes("Percentage"),
Bytes.toBytes(""));

        // Printing the values
        String Question = Bytes.toString(value);
        String SegmentType = Bytes.toString(value1);
        String  SegmentDescription=
Bytes.toString(value2);
        String  Answer= Bytes.toString(value3);
        String  Count= Bytes.toString(value4);
        String  Percentage= Bytes.toString(value5);

        HTable table1 = new
HTable(HBaseConfiguration.create(), "dataset5");
        Scan scan = new Scan();
        scan.setCaching(20);

        scan.addFamily(Bytes.toBytes("Question"));
        ResultScanner scanner =
table.getScanner(scan);

        for (Result result1 = scanner.next(); (result
!= null); result = scanner.next()) {
            for (Cell cell : result.listCells()) {
                String qualifier =
Bytes.toString(cell.getQualifierArray(),
cell.getQualifierOffset(),
                        cell.getQualifierLength());
                String values =
Bytes.toString(cell.getValueArray(),
cell.getValueOffset(), cell.getValueLength());
//              System.out.println("Qualifier : " +
qualifier + " : Value : " + values);


                hashFun(values);

            }
        }

        long lEndTime = System.nanoTime();
        long output = lEndTime - lStartTime;
        output = output / 1000000;
        preparedStmt.setLong(2, output);
        preparedStmt.execute();
        conn.close();

        System.out.println("Disconnected!");

    }

    public static int hashFun(String Str1) {
        int index = 0;
        LinkedHashSet<String> lhs = new
LinkedHashSet<String>();
        LinkedHashMap<String, Integer> lhm = new
LinkedHashMap<String, Integer>();


        Integer freq = lhm.get(Str1);
        lhm.put(Str1, freq == null ? 1 : freq + 1);
        lhs.add(Str1);

        return index;

    }
```

## Appendices image:



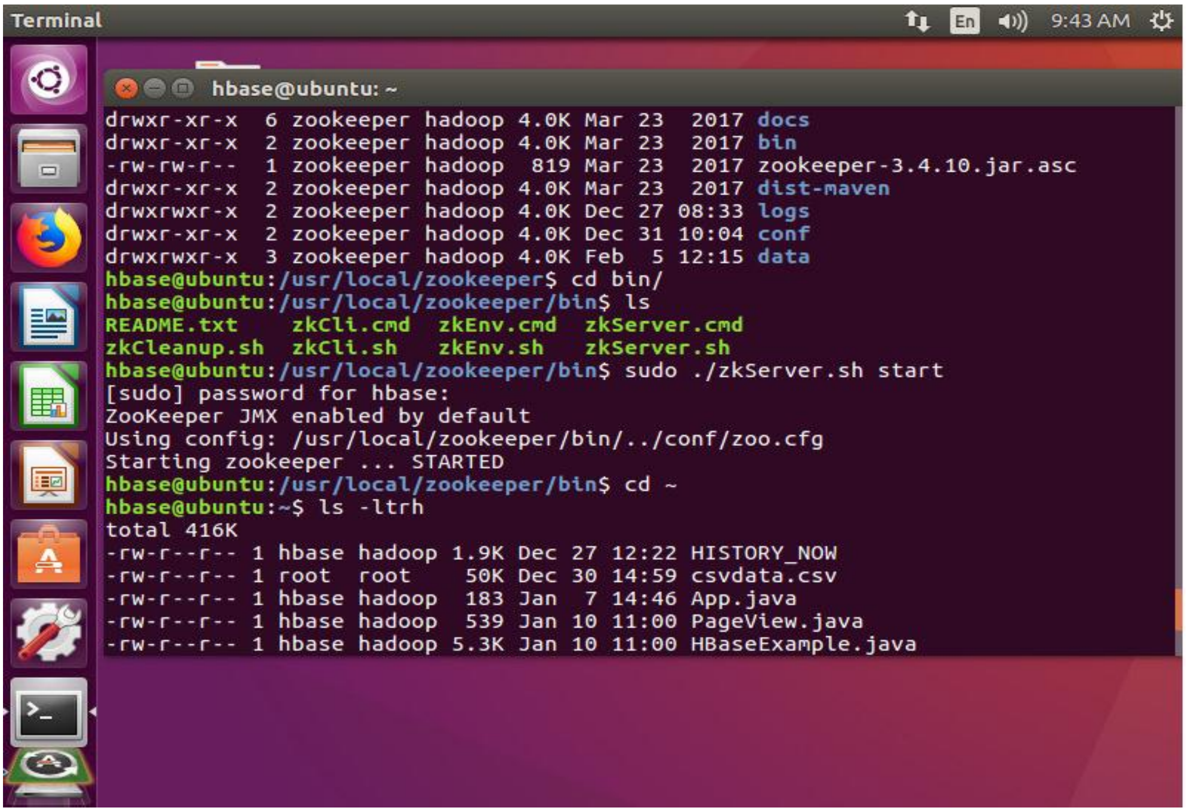
```
+---------------------+
5 rows in set (0.02 sec)

mysql> use nosql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_nosql |
+-----------------+
| exp             |
+-----------------+
1 row in set (0.00 sec)

mysql> desc exp;
+-------+------------------+------+-----+---------+----------------+
| Field | Type             | Null | Key | Default | Extra          |
+-------+------------------+------+-----+---------+----------------+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| start | varchar(32)      | NO   |     | NULL    |                |
| endt  | varchar(32)      | NO   |     | NULL    |                |
+-------+------------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)

mysql> 
```

Figure 1Mysql



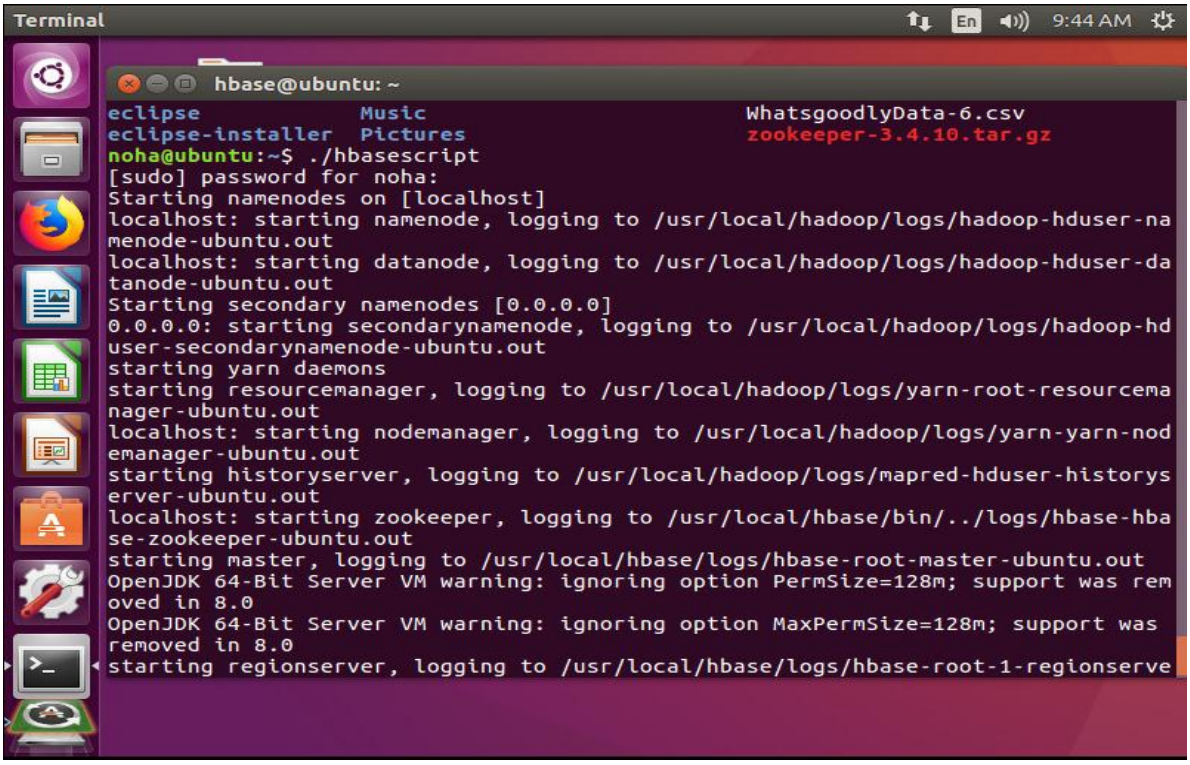
```
  847  ls -ltrh
  848  ./clearcash.sh
  849  cd ~
  850  hbase shell
  851  su noha
  852  hbase shell
  853  nano hashChain.java
  854  cd /etc/
  855  ls -ltrh
  856  history
hbase@ubuntu:/etc$ cd /usr/local/zookeeper/
hbase@ubuntu:/usr/local/zookeeper$ ls -ltrh
total 1.6M
drwxr-xr-x 10 zookeeper hadoop 4.0K Mar 23  2017 contrib
drwxr-xr-x  5 zookeeper hadoop 4.0K Mar 23  2017 recipes
-rw-rw-r--  1 zookeeper hadoop 1.6K Mar 23  2017 README.txt
-rw-rw-r--  1 zookeeper hadoop 1.8K Mar 23  2017 README_packaging.txt
-rw-rw-r--  1 zookeeper hadoop 3.1K Mar 23  2017 NOTICE.txt
-rw-rw-r--  1 zookeeper hadoop  12K Mar 23  2017 LICENSE.txt
-rw-rw-r--  1 zookeeper hadoop 5.6K Mar 23  2017 ivy.xml
-rw-rw-r--  1 zookeeper hadoop 1.7K Mar 23  2017 ivysettings.xml
-rw-rw-r--  1 zookeeper hadoop  83K Mar 23  2017 build.xml
-rw-rw-r--  1 zookeeper hadoop   41 Mar 23  2017 zookeeper-3.4.10.jar.sha1
-rw-rw-r--  1 zookeeper hadoop   33 Mar 23  2017 zookeeper-3.4.10.jar.md5
```

**Figure 2 Zookeeper (1)**



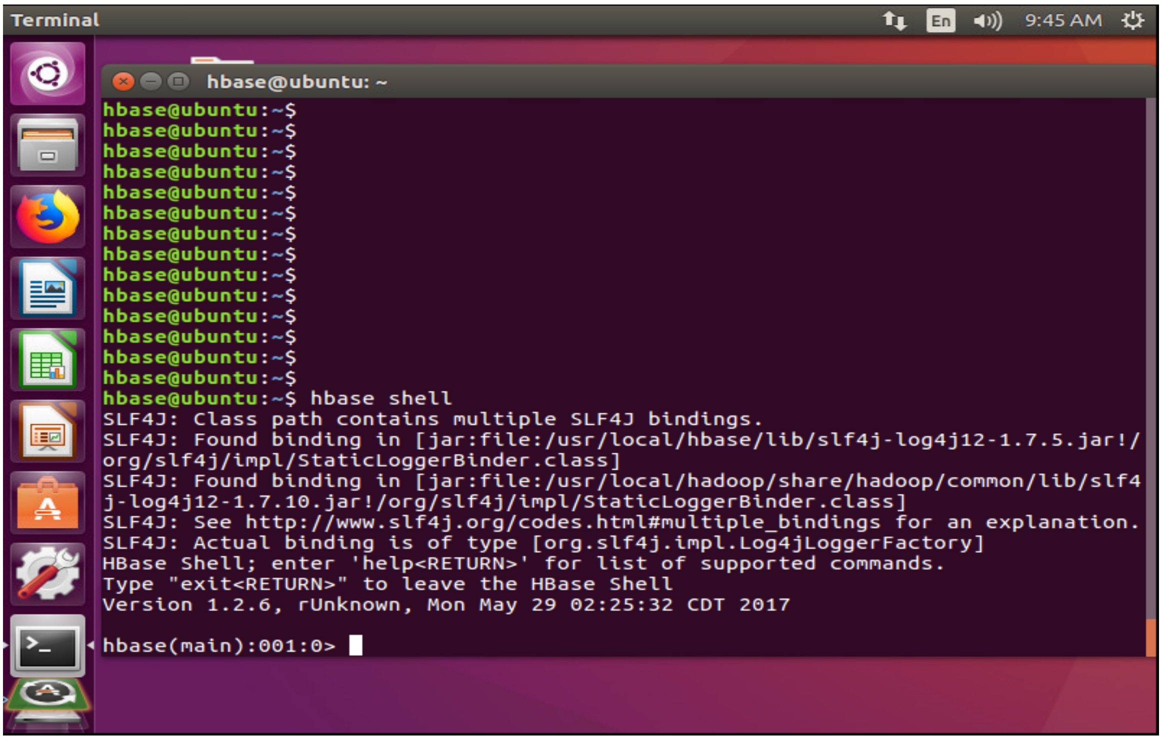
```
drwxr-xr-x  6 zookeeper hadoop 4.0K Mar 23  2017 docs
drwxr-xr-x  2 zookeeper hadoop 4.0K Mar 23  2017 bin
-rw-rw-r--  1 zookeeper hadoop  819 Mar 23  2017 zookeeper-3.4.10.jar.asc
drwxr-xr-x  2 zookeeper hadoop 4.0K Mar 23  2017 dist-maven
drwxrwxr-x  2 zookeeper hadoop 4.0K Dec 27 08:33 logs
drwxr-xr-x  2 zookeeper hadoop 4.0K Dec 31 10:04 conf
drwxrwxr-x  3 zookeeper hadoop 4.0K Feb  5 12:15 data
hbase@ubuntu:/usr/local/zookeeper$ cd bin/
hbase@ubuntu:/usr/local/zookeeper/bin$ ls
README.txt    zkCli.cmd   zkEnv.cmd   zkServer.cmd
zkCleanup.sh  zkCli.sh    zkEnv.sh    zkServer.sh
hbase@ubuntu:/usr/local/zookeeper/bin$ sudo ./zkServer.sh start
[sudo] password for hbase:
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
hbase@ubuntu:/usr/local/zookeeper/bin$ cd ~
hbase@ubuntu:~$ ls -ltrh
total 416K
-rw-r--r-- 1 hbase hadoop 1.9K Dec 27 12:22 HISTORY_NOW
-rw-r--r-- 1 root  root    50K Dec 30 14:59 csvdata.csv
-rw-r--r-- 1 hbase hadoop  183 Jan  7 14:46 App.java
-rw-r--r-- 1 hbase hadoop  539 Jan 10 11:00 PageView.java
-rw-r--r-- 1 hbase hadoop 5.3K Jan 10 11:00 HBaseExample.java
```

**Figure 3 Zookeeper (2)**



```
eclipse              Music                          WhatsgoodlyData-6.csv
eclipse-installer    Pictures                       zookeeper-3.4.10.tar.gz
noha@ubuntu:~$ ./hbasescript
[sudo] password for noha:
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-na
menode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-da
tanode-ubuntu.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-ubuntu.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-root-resourcema
nager-ubuntu.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-yarn-nod
emanager-ubuntu.out
starting historyserver, logging to /usr/local/hadoop/logs/mapred-hduser-historys
erver-ubuntu.out
localhost: starting zookeeper, logging to /usr/local/hbase/bin/../logs/hbase-hba
se-zookeeper-ubuntu.out
starting master, logging to /usr/local/hbase/logs/hbase-root-master-ubuntu.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was rem
oved in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was
removed in 8.0
starting regionserver, logging to /usr/local/hbase/logs/hbase-root-1-regionserve
```

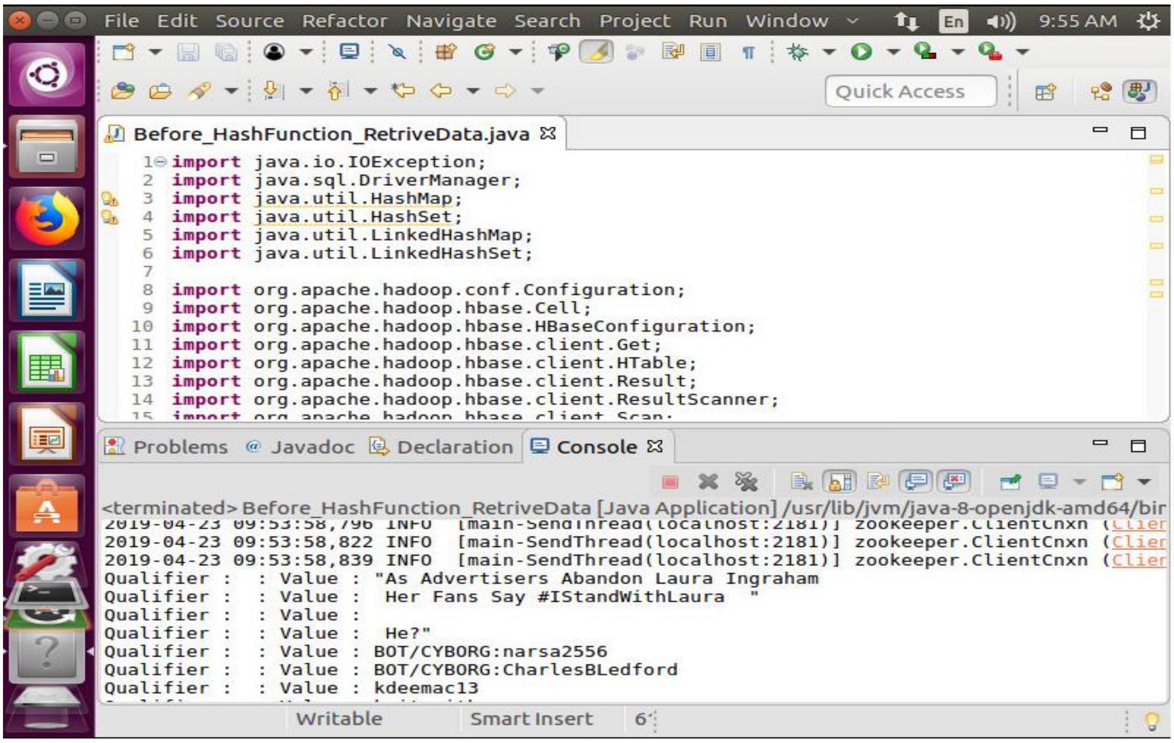
**Figure 2HBase 1**



```
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$
hbase@ubuntu:~$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/slf4j-log4j12-1.7.5.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```

**Figure 3HBase2**

The following appendix shows the time to retrieve data from the database index HBase.

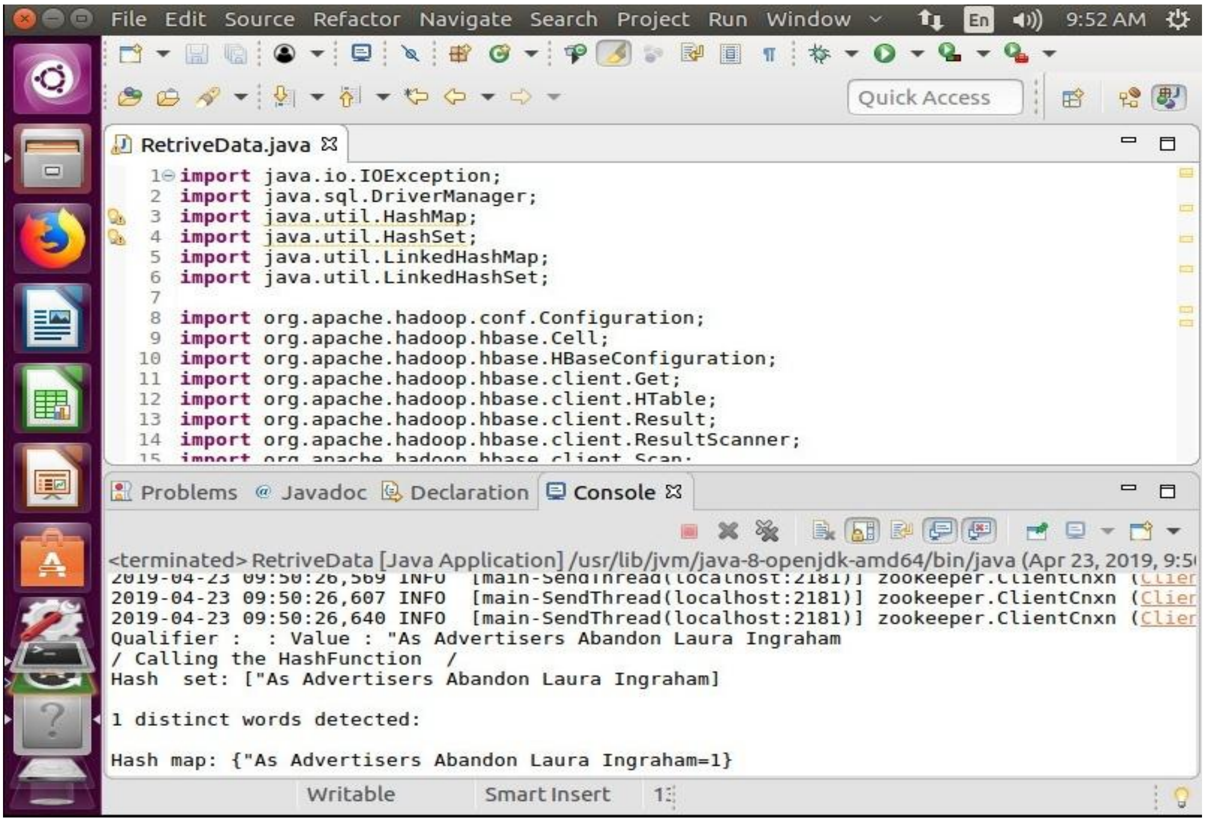


File Edit Source Refactor Navigate Search Project Run Window
9:55 AM
Quick Access
Before_HashFunction_RetriveData.java
Problems
Javadoc
Declaration
Console
Writable
Smart Insert

```
import java.io.IOException;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
```

```
<terminated> Before_HashFunction_RetriveData [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bir
2019-04-23 09:53:58,796 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
2019-04-23 09:53:58,822 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
2019-04-23 09:53:58,839 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
Qualifier :  : Value : "As Advertisers Abandon Laura Ingraham
Qualifier :  : Value :  Her Fans Say #IStandWithLaura  "
Qualifier :  : Value :
Qualifier :  : Value :  He?"
Qualifier :  : Value : BOT/CYBORG:narsa2556
Qualifier :  : Value : BOT/CYBORG:CharlesBLedford
Qualifier :  : Value : kdeemac13
```

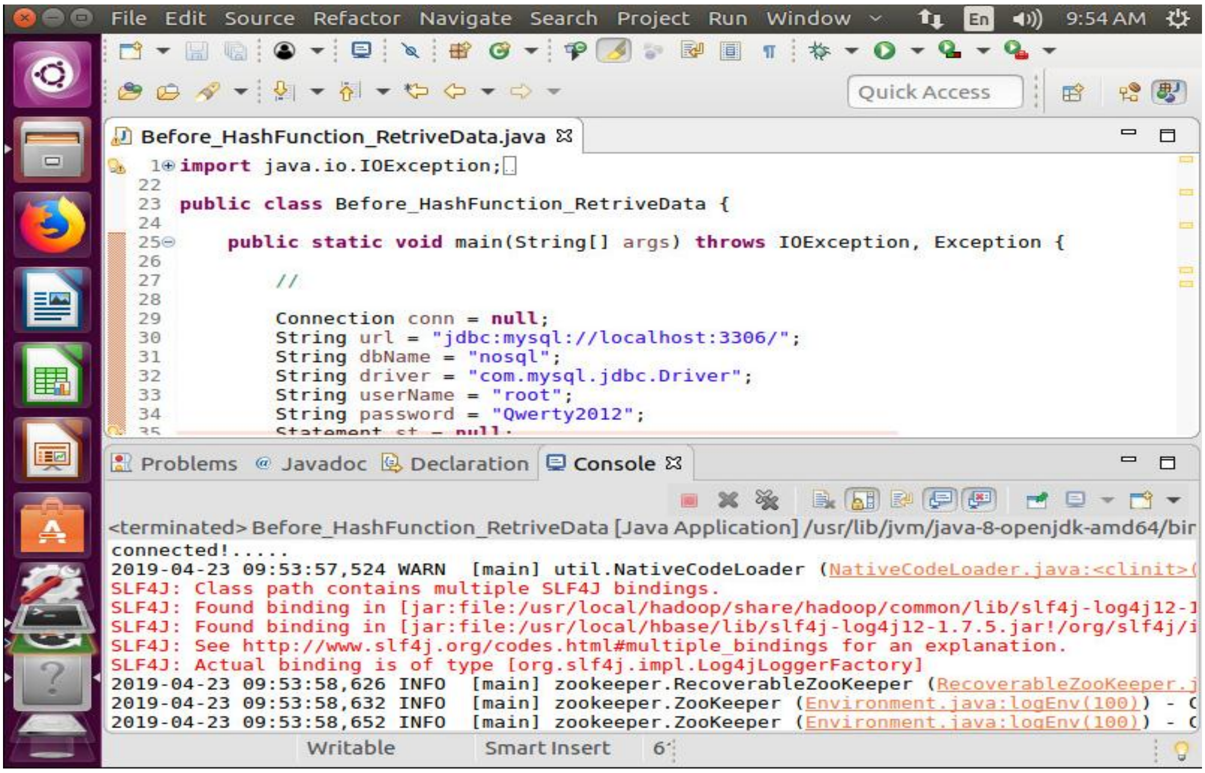
**Figure 4before Hash Function**



```
import java.io.IOException;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
```

```
<terminated> RetriveData [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Apr 23, 2019, 9:5
2019-04-23 09:50:26,569 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
2019-04-23 09:50:26,607 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
2019-04-23 09:50:26,640 INFO  [main-SendThread(localhost:2181)] zookeeper.ClientCnxn (Clier
Qualifier :  : Value : "As Advertisers Abandon Laura Ingraham
/ Calling the HashFunction  /
Hash  set: ["As Advertisers Abandon Laura Ingraham]

1 distinct words detected:

Hash map: {"As Advertisers Abandon Laura Ingraham=1}
```

**Figure 5Retrive data**



```
import java.io.IOException;

public class Before_HashFunction_RetriveData {

    public static void main(String[] args) throws IOException, Exception {

        //

        Connection conn = null;
        String url = "jdbc:mysql://localhost:3306/";
        String dbName = "nosql";
        String driver = "com.mysql.jdbc.Driver";
        String userName = "root";
        String password = "Qwerty2012";
        Statement st = null;
```

```
<terminated> Before_HashFunction_RetriveData [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bir
connected!.....
2019-04-23 09:53:57,524 WARN  [main] util.NativeCodeLoader (NativeCodeLoader.java:<clinit>(
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/i
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2019-04-23 09:53:58,626 INFO  [main] zookeeper.RecoverableZooKeeper (RecoverableZooKeeper.j
2019-04-23 09:53:58,632 INFO  [main] zookeeper.ZooKeeper (Environment.java:logEnv(100)) - C
2019-04-23 09:53:58,652 INFO  [main] zookeeper.ZooKeeper (Environment.java:logEnv(100)) - C
```

**Figure 6before Hash Function**

The following appendices show the time to retrieve data from the database after using the hash algorithm.