Database Management System Project Report

Faculty: Prof. Swathi J.N
Course name: Database Management Systems
Course Code: CSE2004

Project Title:
Comparison and performance analysis of SQL and NoSQL databases

Group Members:

S. No.	Name	Reg. No.
1	Aditya Firoda	16BCE2184
2	Anish yadav	16BCI0177
3	Debayan Bhattacharya	16BCI0199

Certificate

This is to certify that the project work entitled “Comparison and performance analysis of SQL and NoSQL databases” that is being submitted for Database Management System (CSE2004) is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Signature of student:

Aditya Firoda (16BCE2184) -
Anish Yadav (16BCI0177) -
Debayan Bhattacharya (16BCI0199) -

Signature of faculty:

Prof. Swathi J.N -

Acknowledgement

We would like to extend our sincere gratitude to Prof. Swathi J.N for her continued support, inputs, and guidance. The combined team work helped us in achieving our objective and renders our project successful.

Also, we would like to thank the management of VIT University and to the Dean of SCOPE, for providing us with this wonderful opportunity to learn new things, and to expand our horizons, through the ample amount of facilities and opportunities available here in VIT University.

Abstract

The popularity of NoSQL databases has increased due to the need of (1) processing vast amount of data faster than the relational database management systems by taking the advantage of highly scalable architecture, (2) flexible (schema-free) data structure, and (3) low latency and high performance there exists over 225 NoSQL databases that provide different features and characteristics. So it is necessary to reveal which one provides better performance for different data operations. In this project, we experiment the widely used in-memory database to measure their performance in terms of :-

(1) Time taken to complete operations.

(2) Efficiency of memory usage during operations.

Thus the project will help in understanding the deep laying benefits of the different type of databases irrespective of their fundamental nature and quickly visualising the basic pros and cons of the different databases we use in our daily systems at work and other places.

Introduction

Traditional database systems for storage have been based on the relational model. These are widely known as SQL databases named after the language they were queried by . In the last few years, however, non-relational databases have dramatically risen in popularity. These databases are commonly known as NoSQL databases, clearly marking them different from the traditional SQL databases. Most of these are based on storing simple key-value pairs on the premise that simplicity leads to speed.

With the increase in accessibility of Internet and the availability of cheap storage, huge amounts of structured, semistructured,

and unstructured data are captured and stored for a variety of applications. Such data is commonly referred to as Big Data. Processing such vast amount of data requires speed, flexible schemas, and distributed (i.e. noncentralized) databases. NoSQL databases became the preferred currency for operating Big Data they claim to satisfy these requirements. This also lead to a surge in the number of NoSQL database offerings. There are several commercial and open-source implementations of NoSQL databases (such as BigTable and HBase). The large number of NoSQL offerings then leads to questions on differences between these offerings and their suitability in particular applications.

The focus of our project is to compare the key-value stores implementations on NoSQL and SQL databases. While NoSQL databases are generally designed for optimized keyvalue stores, SQL databases are not. Yet, our findings suggest that not all NoSQL databases perform better than SQL databases. We compare read, write, delete, and instantiate operations on the key-value storage. We observe that even within NoSQL databases there is a wide variation in the performance of these operations. We also observe little correlation between performance and the data model each database uses.
The following are the main differences between SQL and NoSQL database management systems as seen from a typical user point of view, starting from the system requirements, the operating systems we can install them on, syntax differences, drivers, to differences of performance regarding query time, insert time and update time on an identical test database.

We choose MongoDB as a NoSQL database management system because it is relatively new on the database market and it is used in many important projects and products, such as: MTV Networks, Craiglist, Disney Interactive Media

Group, Sourceforge, Wordnik, Yabblr, SecondMarket, The Guardian, Forbes, The New York Times, bit.ly, GitHub, FourSquare, Disqus, PiCloud etc.
On the other hand oracle database is an object-relational database management system. An Oracle database system identified by an alphanumeric system identifier comprises at least one instance of the application, along with data storage. An instance—identified persistently by an instantiation number comprises a set of operating-system processes and memory-structures that interact with the storage.

used in-memory database to measure their performance in terms of
(1) the time taken to complete operations, and
(2) how efficiently they use memory during operations.
Thus the project will help in understanding the deep laying benefits of the different
type of databases irrespective of their fundamental nature and quickly visualising the
basic pros and cons of the different databases we use in our daily systems at work and
other places.

Related Research Papers -

1-**MongoDB vs Oracle – database comparison**

Authors-
Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin
Faculty of Automatic Control and Computer Science
Politehnica University of Bucharest
Bucharest, Romania

Abstract
This paper presents the differences between Mongo DB and the old Oracle Database Management system, it reasons out the need for comparing of new Mongo DB as why it should be compared and why it is getting popular.This papers compares the DBMS on the basis of their INSERT, DELETE, UPDATE of tables and time taken to perform these queries.

Finally it shows that Mongo DB is better option if you want just a simple and fast DBMS and still focuses on the importance of SQL because of it's mapping of different relations and performing more complex queries which is not possible in case of MongoDB.

2.**NoSQL Evaluation**

A Use Case Oriented Survey

Authors-
Robin Hecht
Stefan Jablonski

Abstract
"Use the right tool for the Job" is the ideology of this paper and it talks about the various No-SQL databases which are now required in the present scenario where in case of web application such as blogs and services provided by companies like Amazon which offers it services 99.9% time of the year the obsolete Oracle SQL database cannot provide as it does not support integrated failover mechanism. This paper talks about how things data model are visualized in case of NoSQL databases,

-How Key values are stored
-Document Stores instead of antiquated Tables for various types of files
-Column Family Stores which are inspired by the Google BigTable
This Paper also talks about different type of database called Graph database. Eg –
Neo4j and GraphDB
Finally this paper emphasizes on Query possibilities, Concurrency Control, Partitioning and replication and consistency of the data across various platforms.

3. **Database Research: Are We At A Crossroad?**
Reflection on NoSQL
Authors-
Maria Indrawan-Santiago
Faculty of Information Technology
Monash University
Melbourne, Australia
Abstract
This paper talks about the post Relational database era introduced by E.F. Codd and talks about how that NoSQL fans tell that this is the future of database but at the same time admit that the original SQL still has some things which the “Newer” NoSQL
database are not able to copy such as integrity constraints.
But the NoSQL movement has still gained momentum because
-data warehousing require complex operation and read – focus processing.
-Big – Data
- Data as Profit Maker
- Mixed – Structured Data
- Architectural Shift in Computing

And also due to the reason that the NoSQL database are designed around distributed processing and horizontal scalability.

5- **Survey on NoSQL Database**
Authors-
Jing Han, Haihong E, Guan Le
Jian Du
Abstract

This paper shows the need for such Database especially in large scale and high- concurrency applications and also classifies the databases on the basis of the CAP theorem which is (Consistency, Availability, Partition Tolerance)
It also take a brief look at the popular NoSQL databases in use such as
Key Value databases-
Redis, Flare, Tokyo Cabinet – Tokyo Tyrant(not so popular)
Column – Oriented Database
Cassandra, hypertable
#Document – Oriented Database
MongoDB, Couch DB

6- **A performance comparison of SQL and NoSQL databases**
Authors-
Yishan Li and Sathiamoorthy Manoharan
Department of Computer Science
University of Auckland
Abstract
This talks about the first of the Google database that is BigTable which was based on three keys row, column and timestamp.
Also it talks about similar research papers such as one by Tiwari which compares the
databases on the basis of following parameters -

· Scalability
· Consistency
· Support for data models
· Support for queries, and
· Management tools

Later it sets the experimental setup and distinguishes various databases on the basis of the Insert, Update and Delete.
And finally concludes about the future and scope of the various databases for the current needs and future needs as the data explosion occurs around the world and their is a need for making sense of that data.

7-**Type of NoSQL Databases and its Comparison with Relational Databases**

Authors-
Ameya nayak
Anil poriya
Dikshay poojary

Abstract-
NoSQL databases does not completely SQL but compliments it in such a way that they can co-exist. Topics discussed in this paper include data model of NoSQL, query language and it's advantages and disadvantages over RDBMS.

8- **Comparisons of relational databases with Big Data: a teaching Approach**

Authors-
Ali salehnia

Abstract-
Objective of this paper is to provide classification, characteristics and evaluation of available database systems which can be used in Big Data Predictions and analytics.Along

with this the paper also compares the structured, semi-structured and unstructured data as well as dealing with security issues related to these formats.

9-**Comparative analysis of NoSQL (MongoDB) with MySQL Datab**ase

Authors-
Lokesh kumar
Dr. Shalini rajawat
Krati joshi

Abstract-
New and more demanding requirements are arising in databases where we have higher operation speed, good development and higher volumes of data. This paper focuses on using NoSQL to replace the relational database. Also, a method is suggested to integrate with different-2 technology of these two types of database by adding a middleware (Metadata) between application layer and database layer.

10-Comparative Performance Analysis of MySQL and SQL Server Relational
Database
Management Systems in Windows Environment
Authors-
Amlanjyoti saika
Sherin joy
Dhondup dolma
Roseline marry

Abstract-
The huge amount of data flow has made relation database management system one of the most important and popular tools for persistence of data. In this paper an attempthas been made to

set a benchmark in comparing the performance of MySQL against SQL server in windows environment.

11-A Critical Comparison of NoSQL Databases in the Context of Acid and Base

Author-

Deepak GC

Abstract-

This paper focuses on two major types of databases- Relational and NoSQL and further analyse the different models used by these databases. databases. In particular, it will focus on the choice of the ACID or BASE model to be more appropriate for the NoSQL databases. This paper
will also explore some of the approaches and explain why NoSQL database cannot simply follow ACID model.

12-A Comparative Study on the Performance of the Top DBMS Systems

Author-

Youssef bassil

Abstract-

Database systems are the most reliable mean to organise data into collection that searched and updated. There are several database systems available in the market each having their pros and cons in terms of different factors such as reliability, usability, security and performance. This paper presents a comparative study on the performance of the top DBMS systems.

Problem Statement

In this new age of technology where the databases around the world are increasing at tremendous rates, there is a need for new database management and it need is fulfilled by No-SQL database system which moves away from the traditional Relational Database

We take a look at some of these databases and compare their performance with MySQL and check which database system would be much favorable in various situations.

Methodology

In such cases when we need to compare several different databases which follow different architecture and different storage techniques we compare them on the basis of the following parameters we compare

- Indexing
- Sharding
- Transaction Model Followed by them
- Data Structure Implemented in storing them
- Data Model of the Database Management System
- Insertion Times
- Deletion Times
- Updation Times
- Ease of Scalability
- Hashing technique
- ACID or BASE
- Ad-Hoc Query Support
- The best situation where this database can work.

SQL-

Indexing:-

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. Exceptions: Indexes on spatial data types use R-trees; MEMORY tables also

support hash indexes; InnoDB uses inverted lists for FULLTEXT indexes.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in MEMORYtables) are described in Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

MySQL uses indexes for these operations:

- To find the rows matching a WHERE clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on (col1, col2, col3), you have indexed search capabilities on (col1), (col1, col2), and (col1, col2, col3). For more information, see Section 8.3.5, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. For example, VARCHAR(10) and CHAR(10) are the same size, but VARCHAR(10) and CHAR(15) are not.
- For comparisons between nonbinary string columns, both columns should use the same character set. For example, comparing a utf8 column with a latin1 column precludes use of an index.
- Comparison of dissimilar columns (comparing a string column to a temporal or numeric column, for example) may prevent use of indexes if values cannot be compared directly without conversion. For a given value such as 1 in the numeric column, it might compare equal to any number of

values in the string column such as '1', ' 1', '00001', or '01.e1'. This rules out use of any indexes for the string column.

Sharding-

Sharding is a type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data shards.

MySQL Cluster automatically shards (partitions) tables across nodes, enabling databases to scale horizontally on low cost, commodity hardware to serve read and write-intensive workloads, accessed both from SQL and directly via NoSQL APIs. Sharding is entirely transparent to the application which is able to connect to any node in the cluster and have queries automatically access the correct shards.

With its active/active, multi-master architecture, updates can be handled by any node, and are instantly available to all of the other clients accessing the cluster.

Transactions-

A transaction is a set of operations performed so all operations are guaranteed to succeed or fail as one unit.

We should use transactions when several operations must succeed or fail as a unit. The following are some frequent scenarios where use of transactions is recommended:

- In batch processing, where multiple rows must be inserted, updated, or deleted as a single unit.
- Whenever a change to one table requires that other tables be kept consistent.
- When modifying data in two or more databases concurrently.
- In distributed transactions, where data is manipulated in databases on various servers.

Ad-hoc query-

As the word "ad hoc" suggests, this type of query is designed for a "particular purpose,"which is in contrast to a predefined query, which has the same output value on every execution. An ad hoc query does not reside in the system for a long time and is created dynamically on demand by the user. It is more efficient to use an ad hoc query in programming as it saves system resources, but, at the same time complex, ad hoc queries (have multiple variables) also challenge the processing speed and runtime memory of the system.

Data structure implemented-

A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. Search algorithm of SQL servers uses the data structures called B-trees. These are special data structure formats that are specifically tuned(high B Tree branching factors) to perform well on magnetic disk hardware, where the most significant time consuming factor is the seek operation(the magnetic head has to move to a diff part of the file).

Property followed-

SQL severs follow ACID properties. ACID (an acronym for Atomicity, Consistency Isolation, Durability) is a concept that Database Professionals generally look for when evaluating databases and application architectures. For a reliable database all these four attributes should be achieved.

- Atomicity - The atomicity property identifies that the transaction is atomic. An atomic transaction is either fully completed, or is not begun at all. Any updates that a transaction might affect on a system are completed in their entirety. If for any reason an error occurs and the transaction

is unable to complete all of its steps, the then system is returned to the state it was in before the transaction was started. An example of an atomic transaction is an account transfer transaction. The money is removed from account A then placed into account B. If the system fails after removing the money from account A, then the transaction processing system will put the money back into account A, thus returning the system to its original state. This is known as a rollback.

- Consistency - A transaction enforces consistency in the system state by ensuring that at the end of any transaction the system is in a valid state. If the transaction completes successfully, then all changes to the system will have been properly made, and the system will be in a valid state. If any error occurs in a transaction, then any changes already made will be automatically rolled back. This will return the system to its state before the transaction was started. Since the system was in a consistent state when the transaction was started, it will once again be in a consistent state.
- Isolation - When a transaction runs in isolation, it appears to be the only action that the system is carrying out at one time. If there are two transactions that are both performing the same function and are running at the same time, transaction isolation will ensure that each transaction thinks it has exclusive use of the system. This is important in that as the transaction is being executed, the state of the system may not be consistent. The transaction ensures that the system remains consistent after the transaction ends, but during an individual transaction, this may not be the case. If a transaction was not running in isolation, it could access data from the system that may not be consistent. By providing transaction isolation, this is prevented from happening.
- Durability - A transaction is durable in that once it has been successfully completed, all of the changes it made to the system are permanent. There are safeguards that will prevent

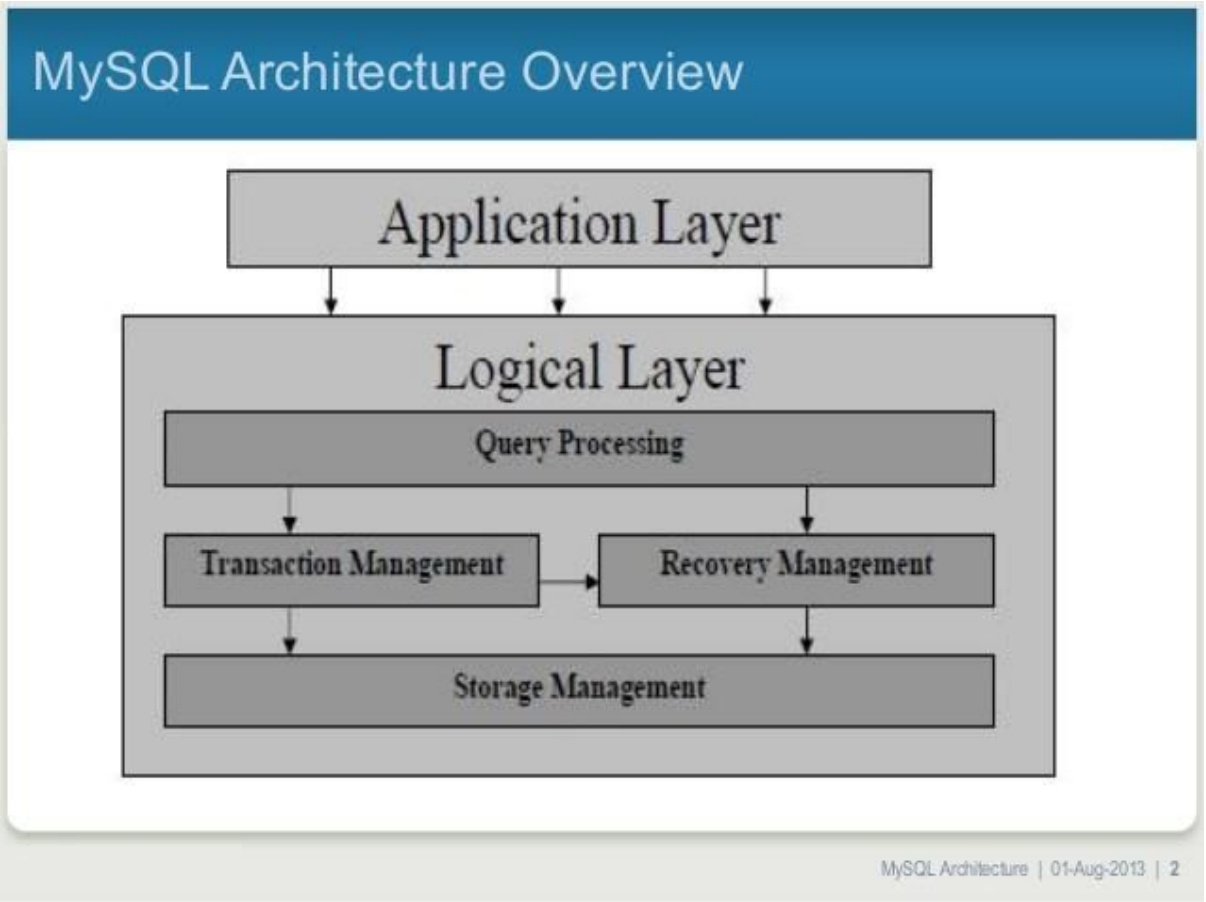
the loss of information, even in the case of system failure. By logging the steps that the transaction performs, the state of the system can be recreated even if the hardware itself has failed. The concept of durability allows the developer to know that a completed transaction is a permanent part of the system, regardless of what happens to the system later on.

Scalability-

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added.

Scaling out a SQL Server environment across multiple systems can be a difficult and complicated project, involving partitioned databases, federation and more. So, when it comes to SQL Server scalability, most organizations prefer to scale individual systems up as much as possible before trying to tackle the out option.

Architecture-



Cassandra-

Apache Cassandra is a free and open-source distributed NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Cassandra offers robust support for clusters spanning multiple data-centers, with asynchronous masterless replication allowing low latency operations for all clients.

Main Features-

Decentralized

Every node in the cluster has the same role. There is no single point of failure. Data is distributed across the cluster (so each

node contains different data), but there is no master as every node can service any request.

Supports replication and multi data-center replication
Replication strategies are configurable. Cassandra is designed as a distributed system, for deployment of large numbers of nodes across multiple data centers. Key features of Cassandra's distributed architecture are specifically tailored for multiple-data center deployment, for redundancy, for failover and disaster recovery.

Scalability-

Designed to have read and write throughput both increase linearly as new machines are added, with the aim of no downtime or interruption to applications.
Fault-tolerant
Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.

Tunable consistency-

Writes and reads offer a tunable level of consistency, all the way from "writes never fail" to "block for all replicas to be readable", with the quorum level in the middle.
MapReduce support
Cassandra has Hadoop integration, with MapReduce support. There is support also for Apache Pig and Apache Hive.
Query language
Cassandra introduced the Cassandra Query Language (CQL). CQL is a simple interface for accessing Cassandra, as an alternative to the traditional Structured Query Language (SQL). CQL adds an abstraction layer that hides implementation details of this structure and provides native syntaxes for collections and other common encodings. Language drivers are available for Java (JDBC), Python (DBAPI2), Node.JS (Helenus), Go (gocql) and C++.

Data Model-
Cassandra is essentially a hybrid between a key-value and a column-oriented (or tabular) database management system. Its data model is a partitioned row store with tunable consistency. Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns may be indexed separately from the primary key.

Tables may be created, dropped, and altered at run-time without blocking updates and queries.

Cassandra cannot do joins or subqueries. Rather, Cassandra emphasizes denormalization through features like collections.

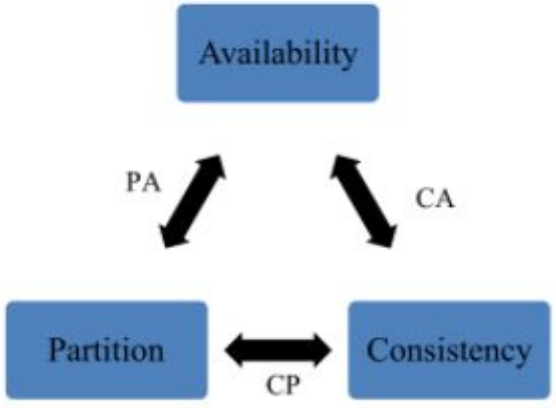
Clustering-
When the cluster for Apache Cassandra is designed, an important point is to select the right partitioner. Two partitioners exist-

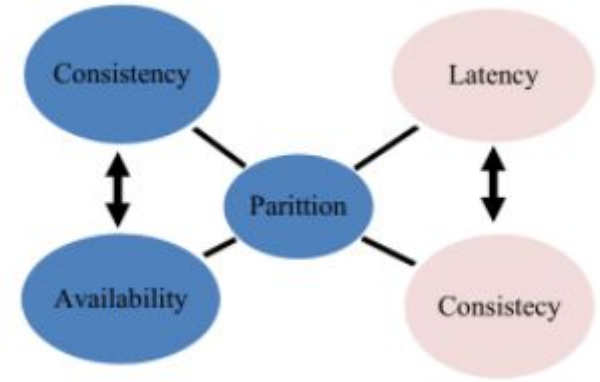
- OrderPreservingPartitioner (OPP) : This partitioner distributes the key-value pairs in a natural way so that similar keys are not far apart. The advantage is that fewer nodes have to be accessed. The drawback is the uneven distribution of the key-value pairs.
- RandomPartitioner (RP) : This partitioner randomly

distributes the key-value pairs over the network, resulting in a good load balancing. Compared to OPP, more nodes have to be accessed to get a number of keys.

Transactions -
Cassandra does not use RDBMS ACID transactions with rollback or locking mechanisms, but instead offers atomic, isolated, and durable transactions with eventual/tunable consistency that lets the user decide how strong or eventual they want each transaction's consistency to be.

As a non-relational database, Cassandra does not support joins or foreign keys, and consequently does not offer consistency in the ACID sense. For example, when moving money from account A to B the total in the accounts does not change. Cassandra supports atomicity and isolation at the row-level, but trades transactional isolation and atomicity for high availability and fast write performance. Cassandra writes are durable.





Atomicity-
In Cassandra, a write operation is atomic at the partition level, meaning the insertions or updates of two or more rows in the same partition are treated as one write operation. A delete operation is also atomic at the partition level.

Isolation-
Cassandra write and delete operations are performed with full row-level isolation. This means that a write to a row within a single partition on a single node is only visible to the client performing the operation – the operation is restricted to this scope until it is complete.

Durability-
Writes in Cassandra are durable. All writes to a replica node are recorded both in memory and in a commit log on disk before they are acknowledged as a success. If a crash or server failure occurs before the memtables are flushed to disk, the commit log is replayed on restart to recover any lost writes.
Data Storage Technique in Cassandra-

For each column family, there are 3 layer of data stores: memtable, commit log and SSTable.

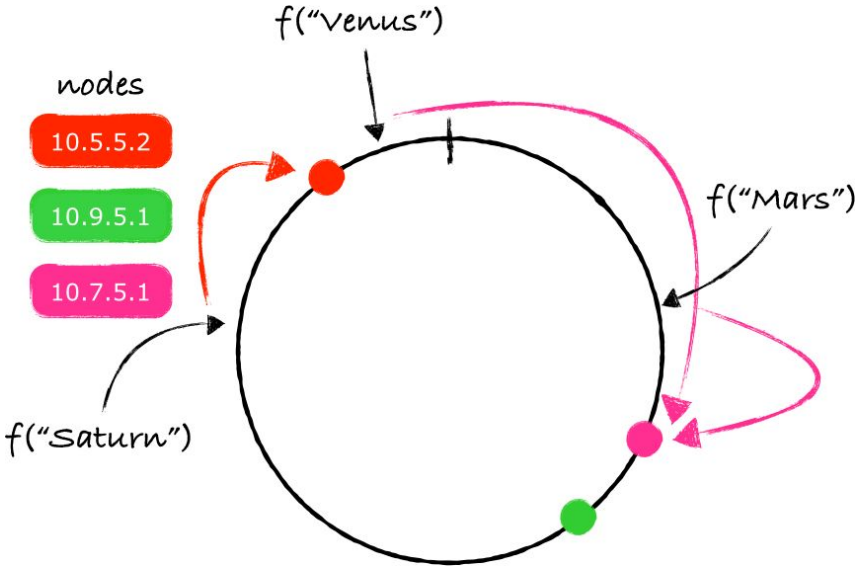
For efficiency, Cassandra does not repeat the names of the columns in memory or in the SSTable.

Compaction -
To improve read performance as well as to utilize disk space, Cassandra periodically does compaction to create & use new consolidated SSTable files instead of multiple old SSTables.
SizeTieredCompactionStrategy-is designed for write-intensive workloads
LeveledCompactionStrategy-for read-intensive workloads

Compaction Algorithm-
*IF ((bucket avg size * bucket_low < SStable' size < bucket avg size * bucket_high) OR (SStable' size < min_sstable_size AND bucket avg size < min_sstable_size))*

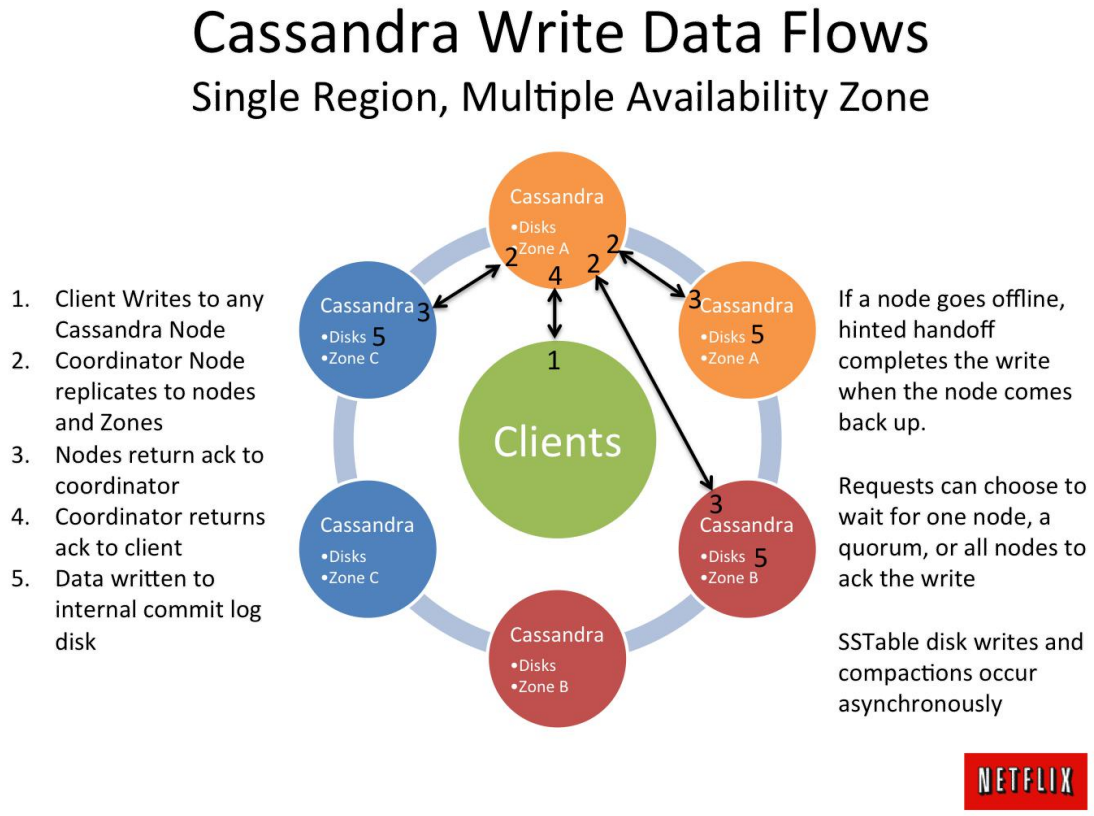
then add the SSTable to the bucket and compute the new avg. size of the bucket.
ELSE create a new bucket with the SSTable.

Consistent hashing-
Consistent hashing allows distribution of data across a cluster to minimize reorganization when nodes are added or removed.
Consistent hashing partitions data based on the partition key.


f("Venus")
nodes
10.5.5.2
10.9.5.1
10.7.5.1
f("Mars")
f("Saturn")

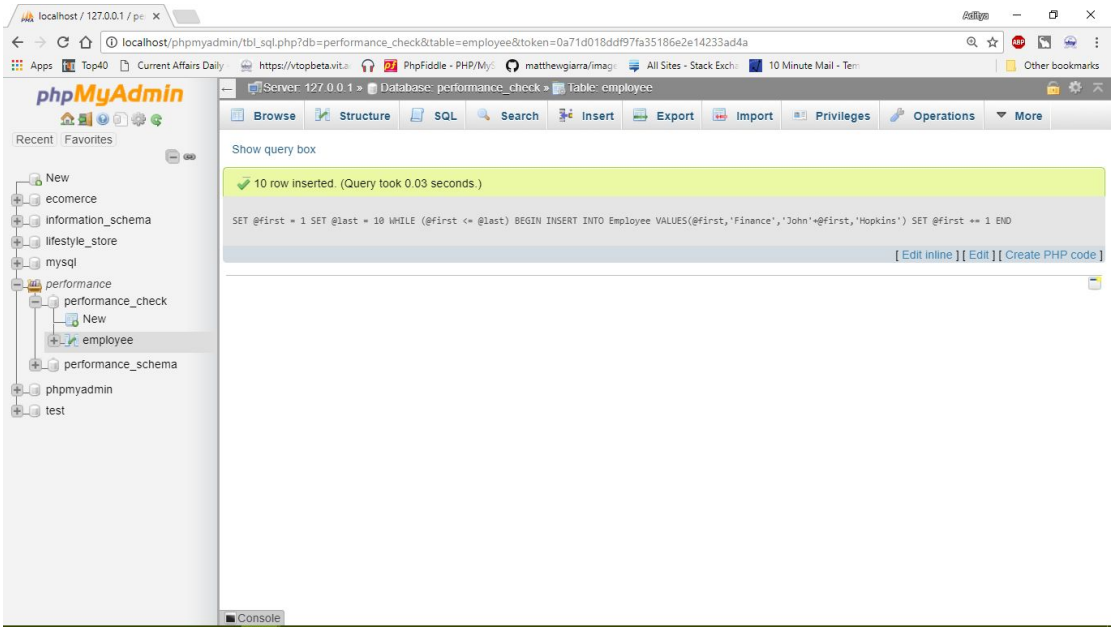
Scalability -
supports both horizontal and vertical scalability

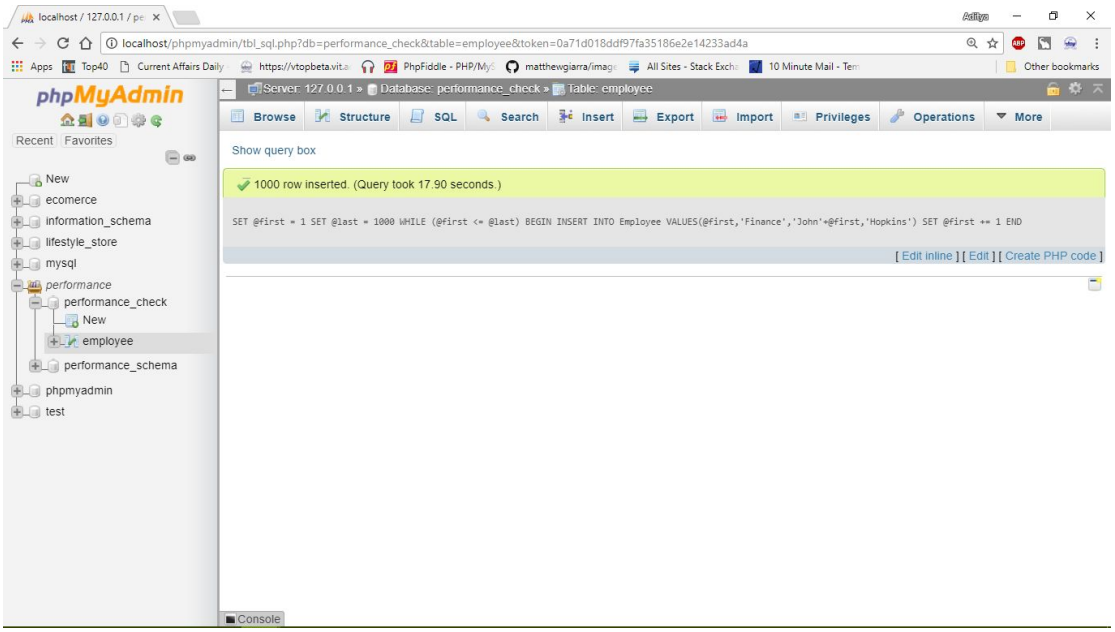
Architecture-

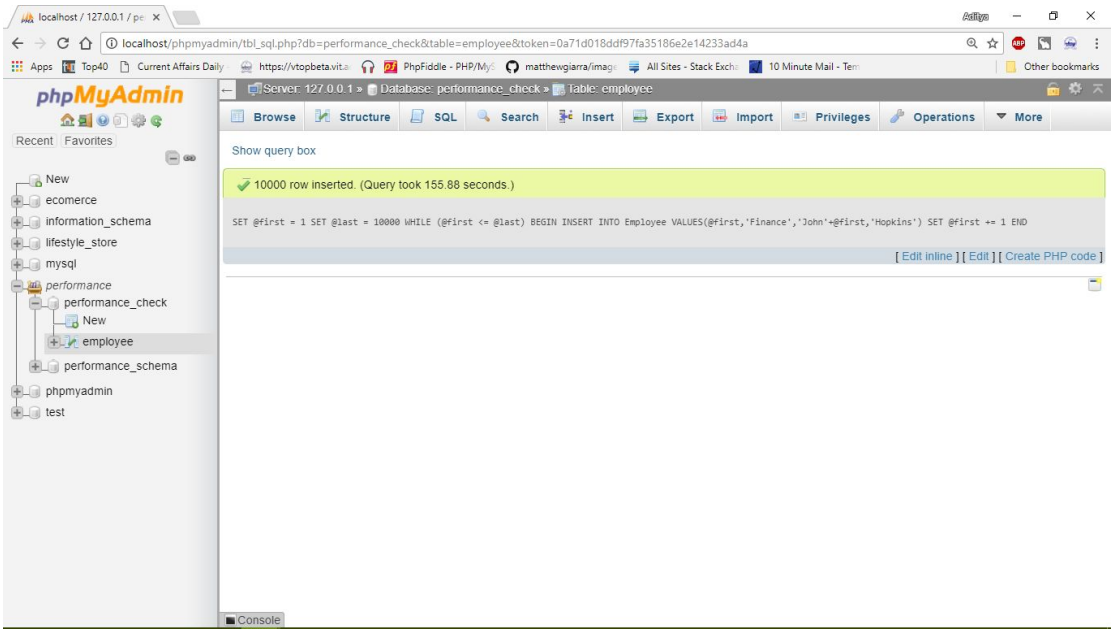


Snapshots-
MySQL

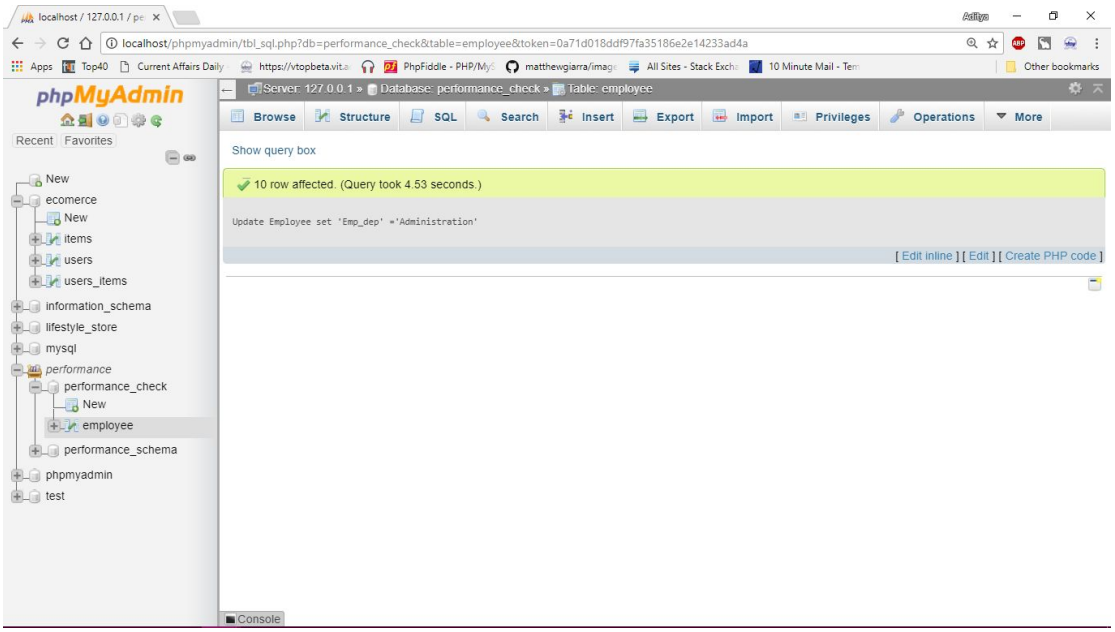
Insertion

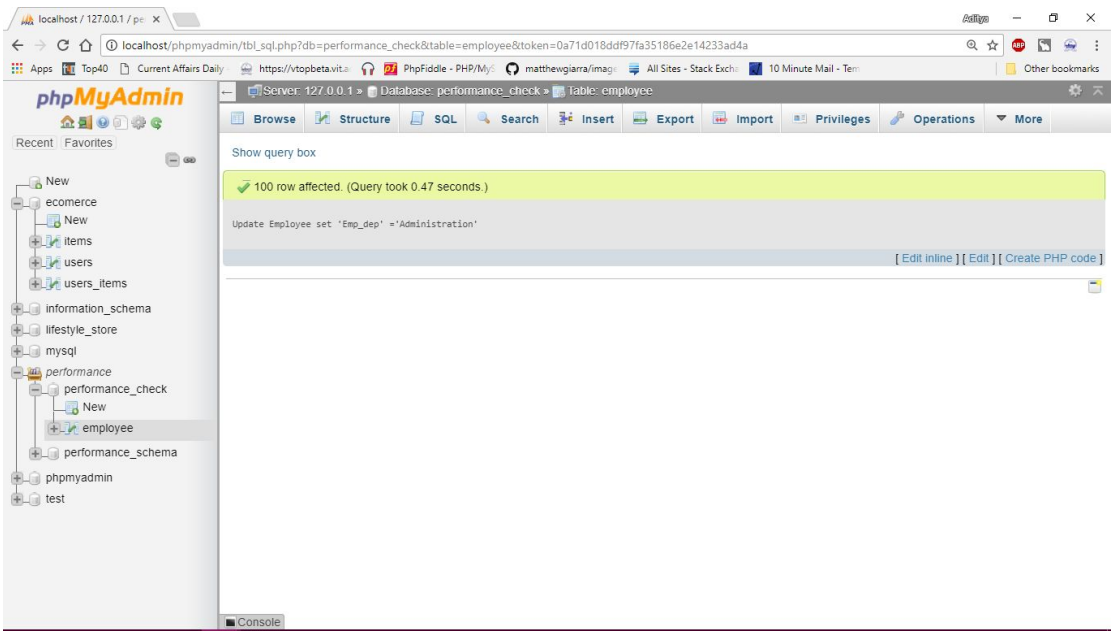

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent
Favorites
New
ecomerce
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
More
Show query box
10 row inserted. (Query took 0.03 seconds.)
SET @first = 1 SET @last = 10 WHILE (@first <= @last) BEGIN INSERT INTO Employee VALUES(@first,'Finance','John'+@first,'Hopkins') SET @first += 1 END
[Edit inline] [Edit] [Create PHP code]
Console

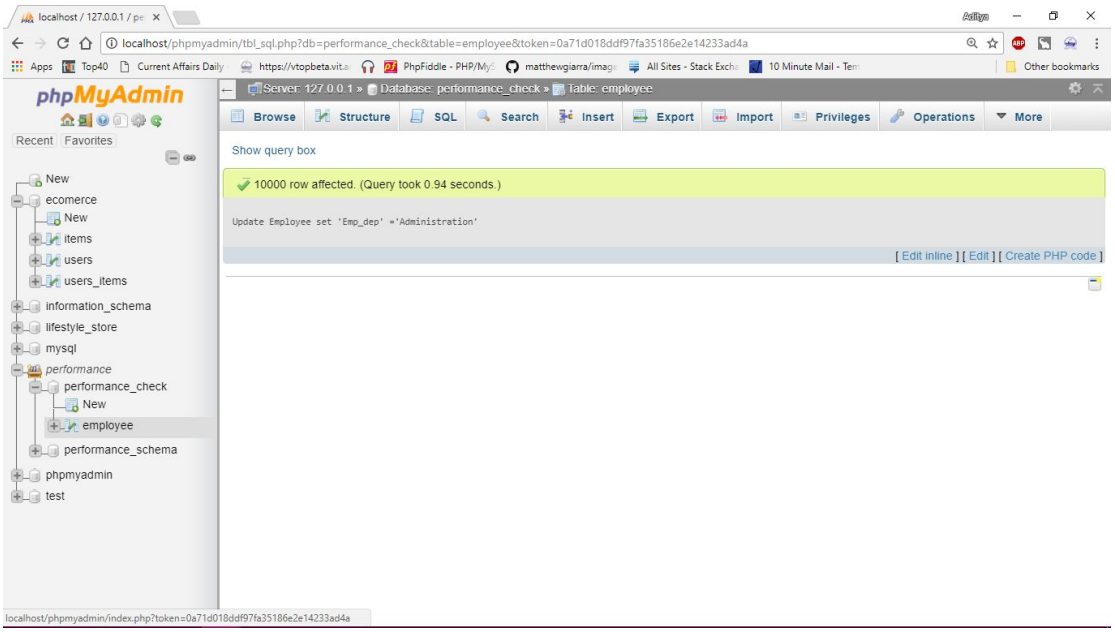

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
1000 row inserted. (Query took 17.90 seconds.)
SET @first = 1 SET @last = 1000 WHILE (@first <= @last) BEGIN INSERT INTO Employee VALUES(@first,'Finance','John'+@first,'Hopkins') SET @first += 1 END
[Edit inline] [Edit] [Create PHP code]
Console

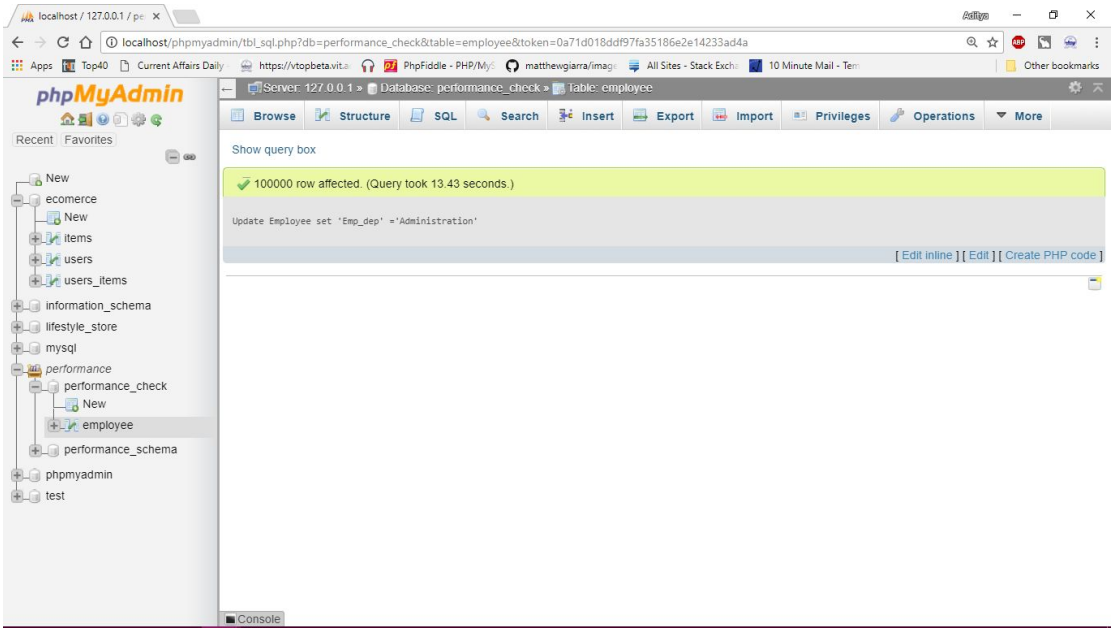

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
10000 row inserted. (Query took 155.88 seconds.)
SET @first = 1 SET @last = 10000 WHILE (@first <= @last) BEGIN INSERT INTO Employee VALUES(@first,'Finance','John'+@first,'Hopkins') SET @first += 1 END
[Edit inline] [Edit] [Create PHP code]
Console

Updation-

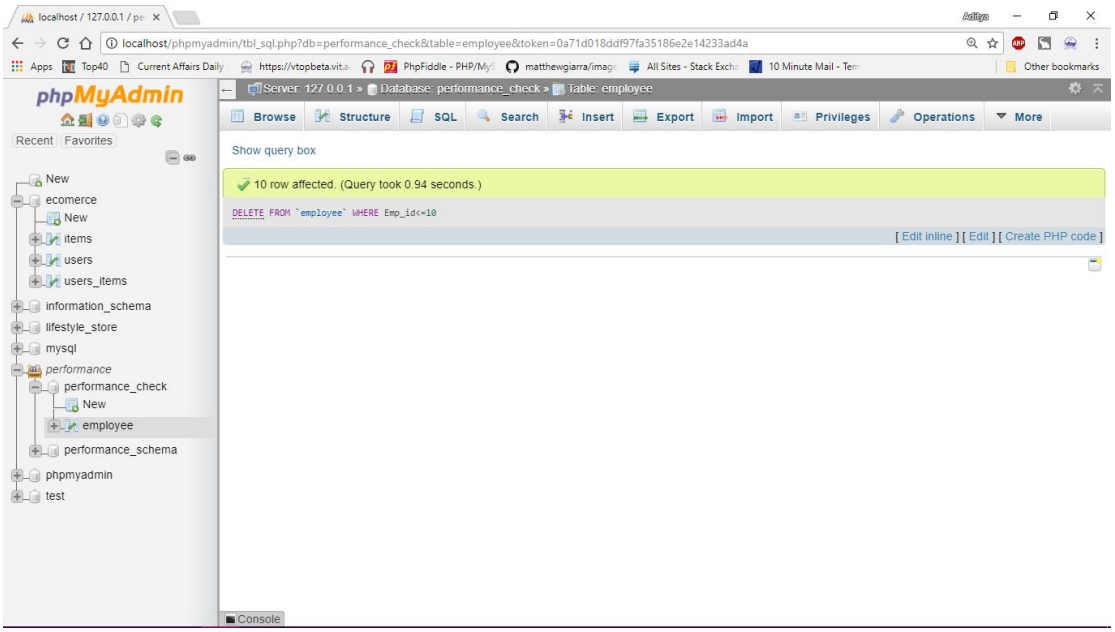

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
10 row affected. (Query took 4.53 seconds.)
Update Employee set 'Emp_dep' ='Administration'
[Edit inline] [Edit] [Create PHP code]
Console

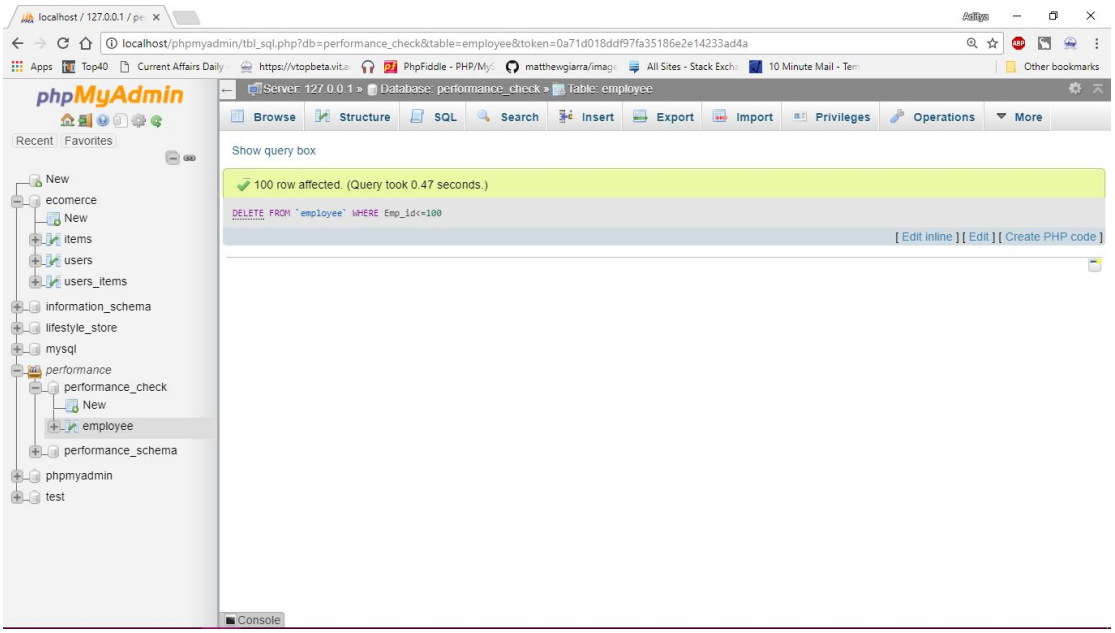

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
100 row affected. (Query took 0.47 seconds.)
Update Employee set 'Emp_dep' ='Administration'
[Edit inline] [Edit] [Create PHP code]
Console

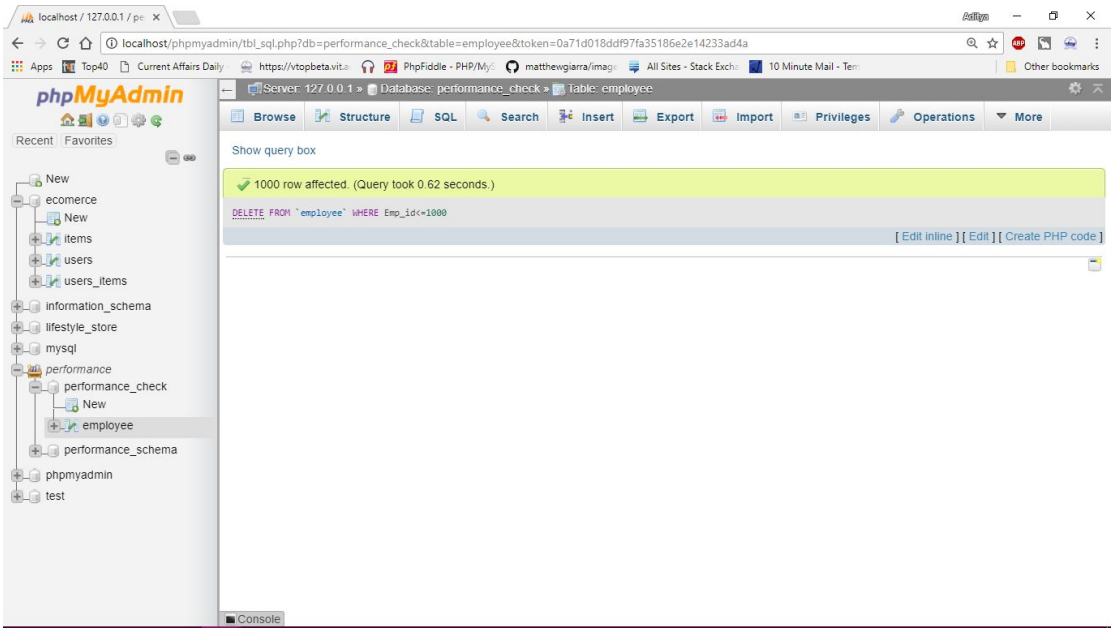

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
10000 row affected. (Query took 0.94 seconds.)
Update Employee set 'Emp_dep' ='Administration'
[Edit inline] [Edit] [Create PHP code]
localhost/phpmyadmin/index.php?token=0a71d018ddf97fa35186e2e14233ad4a

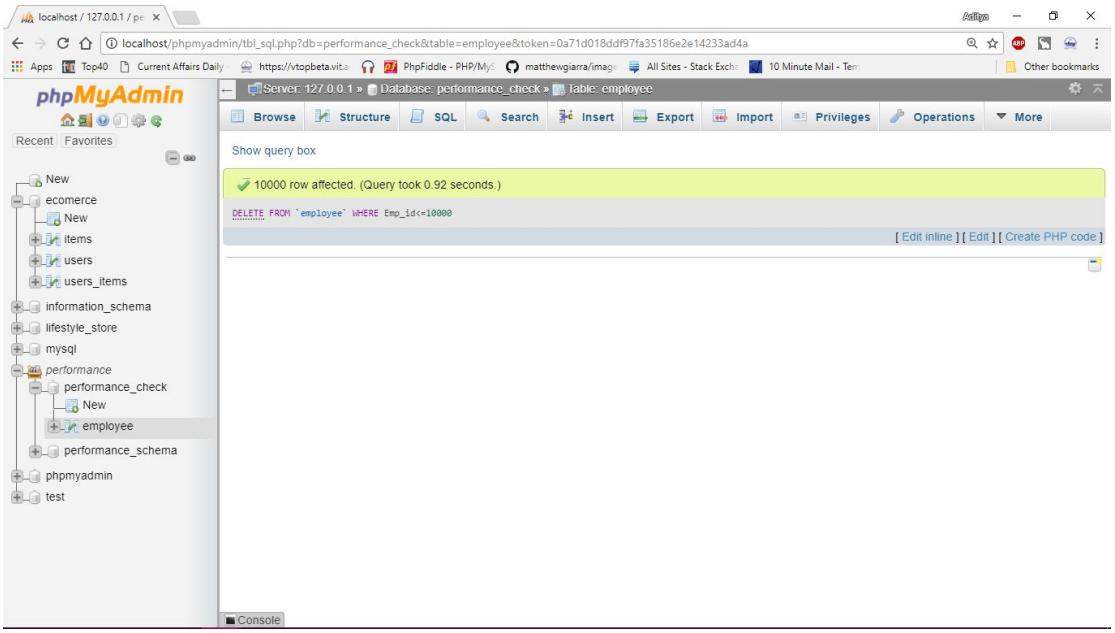

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
100000 row affected. (Query took 13.43 seconds.)
Update Employee set 'Emp_dep' ='Administration'
[Edit inline] [Edit] [Create PHP code]
Console

Deletion-


localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
10 row affected. (Query took 0.94 seconds.)
DELETE FROM `employee` WHERE Emp_id<=10
[Edit inline] [Edit] [Create PHP code]
Console

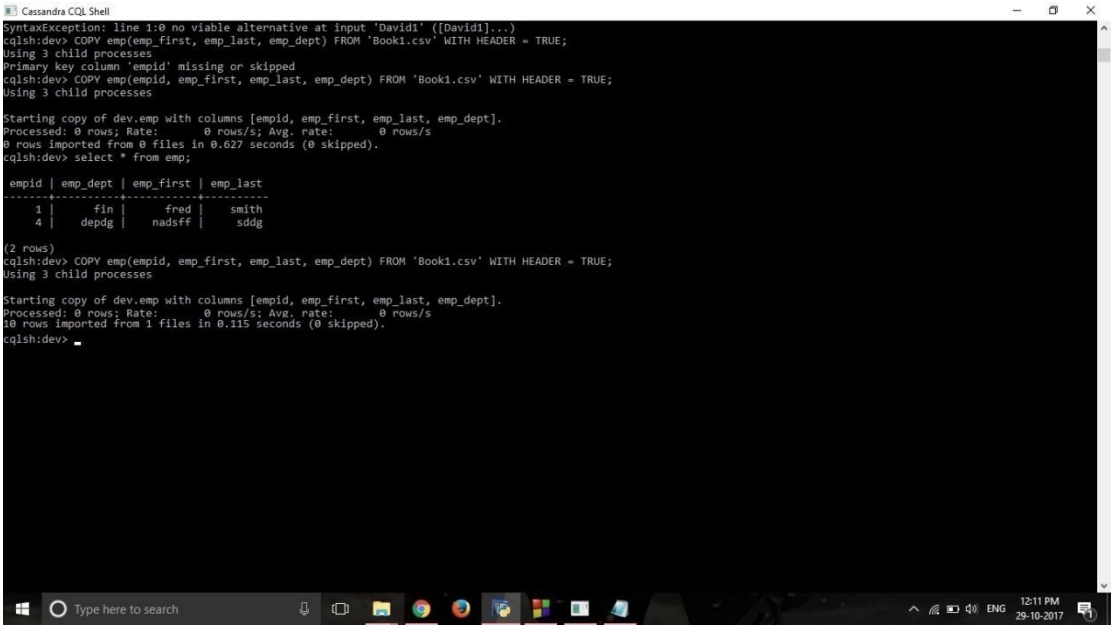

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
100 row affected. (Query took 0.47 seconds.)
DELETE FROM `employee` WHERE Emp_id<=100
[Edit inline] [Edit] [Create PHP code]
Console

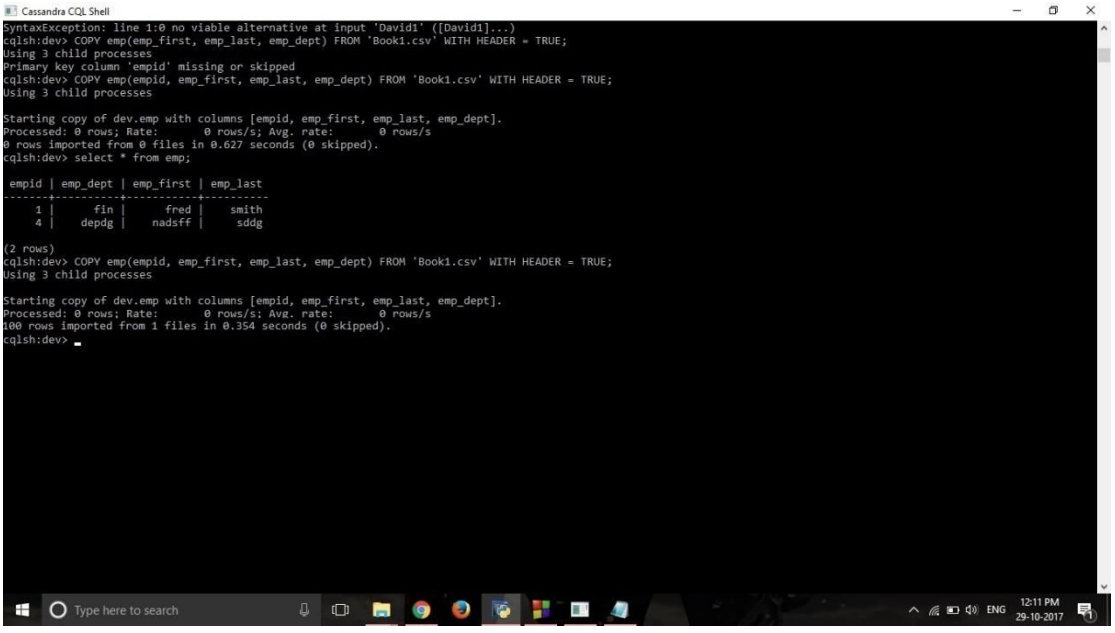

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
1000 row affected. (Query took 0.62 seconds.)
DELETE FROM `employee` WHERE Emp_id<=1000
[Edit inline] [Edit] [Create PHP code]
Console

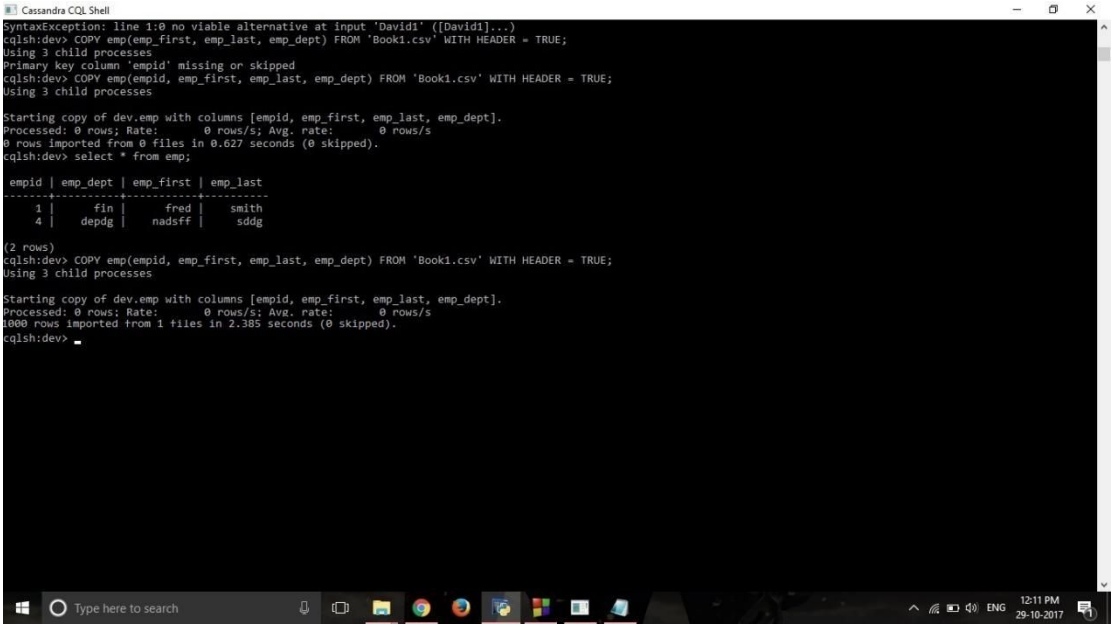

localhost / 127.0.0.1 / pe
localhost/phpmyadmin/tbl_sql.php?db=performance_check&table=employee&token=0a71d018ddf97fa35186e2e14233ad4a
phpMyAdmin
Recent Favorites
New
ecomerce
New
items
users
users_items
information_schema
lifestyle_store
mysql
performance
performance_check
New
employee
performance_schema
phpmyadmin
test
Server: 127.0.0.1 » Database: performance_check » Table: employee
Browse Structure SQL Search Insert Export Import Privileges Operations More
Show query box
10000 row affected. (Query took 0.92 seconds.)
DELETE FROM `employee` WHERE Emp_id<=10000
[Edit inline] [Edit] [Create PHP code]
Console

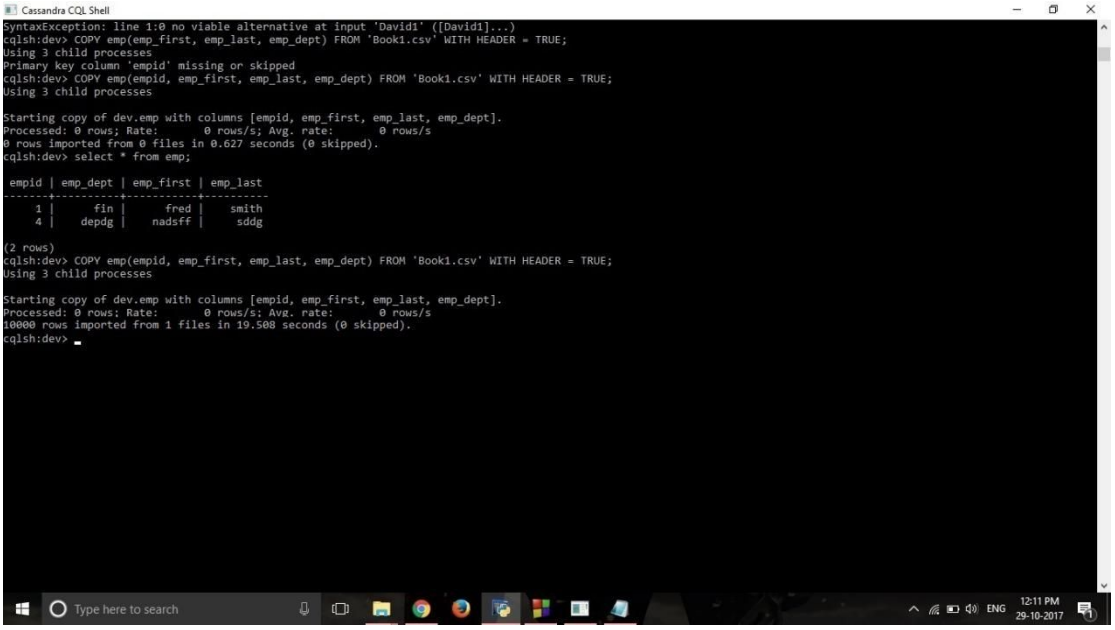
Cassandra-

Insertion-

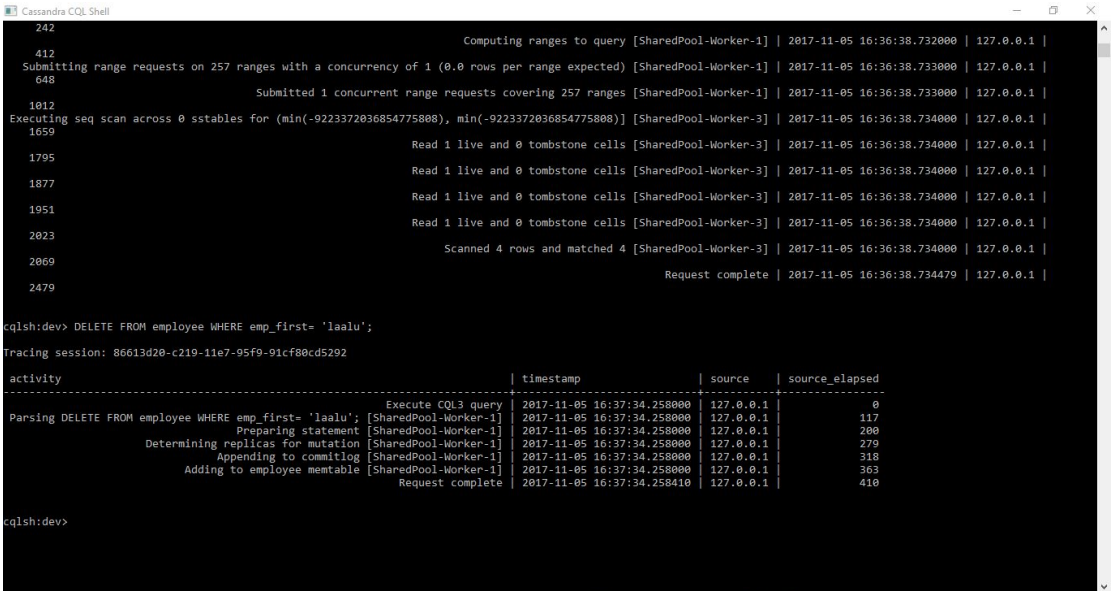


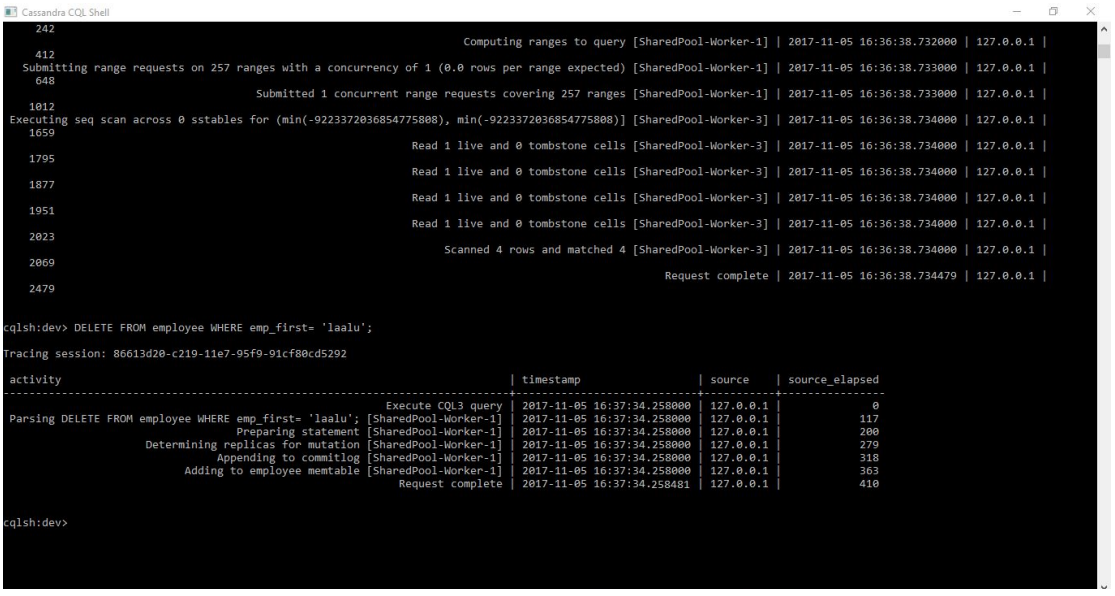


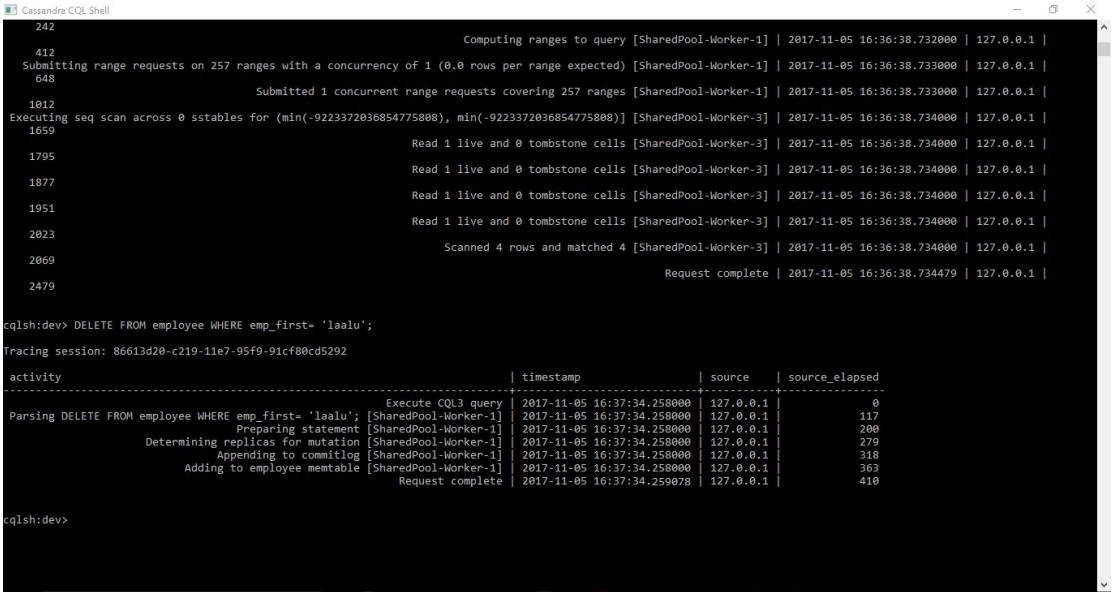

Cassandra CQL Shell
SyntaxException: line 1:0 no viable alternative at input 'David1' ([David1]...)
cqlsh:dev> COPY emp(emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Primary key column 'empid' missing or skipped
cqlsh:dev> COPY emp(empid, emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Starting copy of dev.emp with columns [empid, emp_first, emp_last, emp_dept].
Processed: 0 rows; Rate: 0 rows/s; Avg. rate: 0 rows/s
0 rows imported from 0 files in 0.627 seconds (0 skipped).
cqlsh:dev> select * from emp;
empid | emp_dept | emp_first | emp_last
1 | fin | fred | smith
4 | depdg | nadsff | sddg
(2 rows)
cqlsh:dev> COPY emp(empid, emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Starting copy of dev.emp with columns [empid, emp_first, emp_last, emp_dept].
Processed: 0 rows; Rate: 0 rows/s; Avg. rate: 0 rows/s
1000 rows imported from 1 files in 2.385 seconds (0 skipped).
cqlsh:dev>
Type here to search
ENG
12:11 PM
29-10-2017

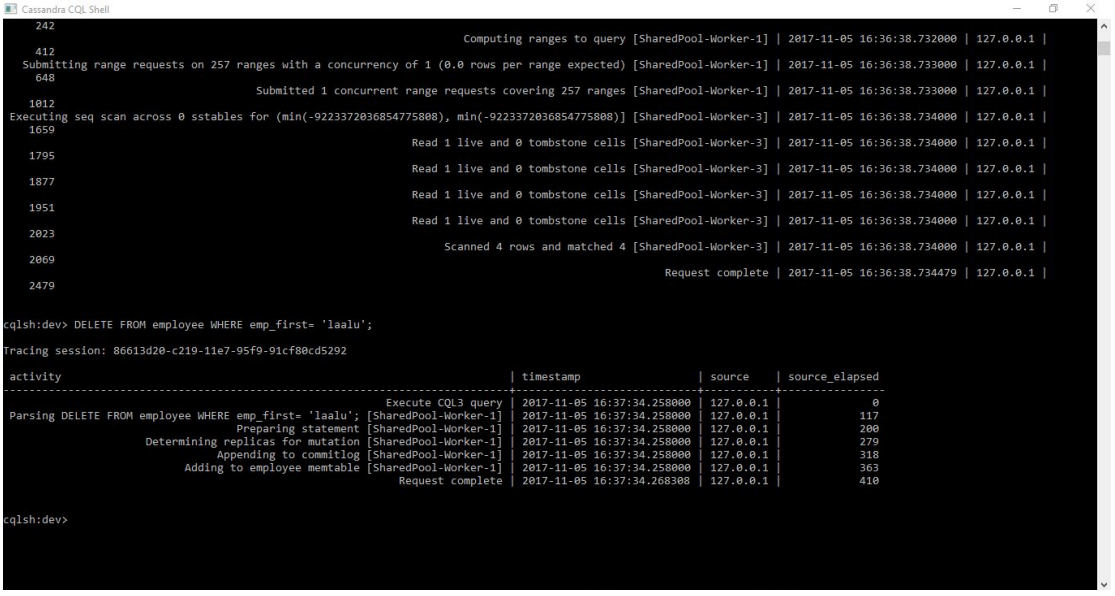

Cassandra CQL Shell
SyntaxException: line 1:0 no viable alternative at input 'David1' ([David1]...)
cqlsh:dev> COPY emp(emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Primary key column 'empid' missing or skipped
cqlsh:dev> COPY emp(empid, emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Starting copy of dev.emp with columns [empid, emp_first, emp_last, emp_dept].
Processed: 0 rows; Rate: 0 rows/s; Avg. rate: 0 rows/s
0 rows imported from 0 files in 0.627 seconds (0 skipped).
cqlsh:dev> select * from emp;
empid | emp_dept | emp_first | emp_last
1 | fin | fred | smith
4 | depdg | nadsff | sddg
(2 rows)
cqlsh:dev> COPY emp(empid, emp_first, emp_last, emp_dept) FROM 'Book1.csv' WITH HEADER = TRUE;
Using 3 child processes
Starting copy of dev.emp with columns [empid, emp_first, emp_last, emp_dept].
Processed: 0 rows; Rate: 0 rows/s; Avg. rate: 0 rows/s
10000 rows imported from 1 files in 19.508 seconds (0 skipped).
cqlsh:dev>
Type here to search
ENG
12:11 PM
29-10-2017

Deletion-


Cassandra CQL Shell
242 | Computing ranges to query [SharedPool-Worker-1] | 2017-11-05 16:36:38.732000 | 127.0.0.1 |
412 | Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
648 | Submitted 1 concurrent range requests covering 257 ranges [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
1012 | Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1659 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1795 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1877 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1951 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2023 | Scanned 4 rows and matched 4 [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2069 | Request complete | 2017-11-05 16:36:38.734479 | 127.0.0.1 |
2479
cqlsh:dev> DELETE FROM employee WHERE emp_first= 'laalu';
Tracing session: 86613d20-c219-11e7-95f9-91cf80cd5292
activity | timestamp | source | source_elapsed
Execute CQL3 query | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 0
Parsing DELETE FROM employee WHERE emp_first= 'laalu'; [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 117
Preparing statement [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 200
Determining replicas for mutation [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 279
Appending to commitlog [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 318
Adding to employee memtable [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 363
Request complete | 2017-11-05 16:37:34.258410 | 127.0.0.1 | 410
cqlsh:dev>


Cassandra CQL Shell
242 | Computing ranges to query [SharedPool-Worker-1] | 2017-11-05 16:36:38.732000 | 127.0.0.1 |
412 | Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
648 | Submitted 1 concurrent range requests covering 257 ranges [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
1012 | Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1659 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1795 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1877 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1951 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2023 | Scanned 4 rows and matched 4 [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2069 | Request complete | 2017-11-05 16:36:38.734479 | 127.0.0.1 |
2479
cqlsh:dev> DELETE FROM employee WHERE emp_first= 'laalu';
Tracing session: 86613d20-c219-11e7-95f9-91cf80cd5292
activity | timestamp | source | source_elapsed
Execute CQL3 query | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 0
Parsing DELETE FROM employee WHERE emp_first= 'laalu'; [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 117
Preparing statement [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 200
Determining replicas for mutation [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 279
Appending to commitlog [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 318
Adding to employee memtable [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 363
Request complete | 2017-11-05 16:37:34.258481 | 127.0.0.1 | 410
cqlsh:dev>


Cassandra CQL Shell
242 | Computing ranges to query [SharedPool-Worker-1] | 2017-11-05 16:36:38.732000 | 127.0.0.1 |
412 | Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
648 | Submitted 1 concurrent range requests covering 257 ranges [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
1012 | Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-9223372036854775808)) [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1659 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1795 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1877 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1951 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2023 | Scanned 4 rows and matched 4 [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2069 | Request complete | 2017-11-05 16:36:38.734479 | 127.0.0.1 |
2479
cqlsh:dev> DELETE FROM employee WHERE emp_first= 'laalu';
Tracing session: 86613d20-c219-11e7-95f9-91cf80cd5292
activity | timestamp | source | source_elapsed
Execute CQL3 query | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 0
Parsing DELETE FROM employee WHERE emp_first= 'laalu'; [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 117
Preparing statement [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 200
Determining replicas for mutation [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 279
Appending to commitlog [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 318
Adding to employee memtable [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 363
Request complete | 2017-11-05 16:37:34.259078 | 127.0.0.1 | 410
cqlsh:dev>


Cassandra CQL Shell
242 | Computing ranges to query [SharedPool-Worker-1] | 2017-11-05 16:36:38.732000 | 127.0.0.1 |
412 | Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
648 | Submitted 1 concurrent range requests covering 257 ranges [SharedPool-Worker-1] | 2017-11-05 16:36:38.733000 | 127.0.0.1 |
1012 | Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-9223372036854775808)) [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1659 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1795 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1877 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
1951 | Read 1 live and 0 tombstone cells [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2023 | Scanned 4 rows and matched 4 [SharedPool-Worker-3] | 2017-11-05 16:36:38.734000 | 127.0.0.1 |
2069 | Request complete | 2017-11-05 16:36:38.734479 | 127.0.0.1 |
2479
cqlsh:dev> DELETE FROM employee WHERE emp_first= 'laalu';
Tracing session: 86613d20-c219-11e7-95f9-91cf80cd5292
activity | timestamp | source | source_elapsed
Execute CQL3 query | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 0
Parsing DELETE FROM employee WHERE emp_first= 'laalu'; [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 117
Preparing statement [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 200
Determining replicas for mutation [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 279
Appending to commitlog [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 318
Adding to employee memtable [SharedPool-Worker-1] | 2017-11-05 16:37:34.258000 | 127.0.0.1 | 363
Request complete | 2017-11-05 16:37:34.268308 | 127.0.0.1 | 410
cqlsh:dev>

Conclusion-

We have explored and compared different types of NoSQL databases. Two main drivers for these databases are the needs of many organizations to process large amounts of data which in some cases has no obvious tabular structure.

While NoSQL databases are generally optimized for key-value stores, SQL databases are not. Yet, we find that not all NoSQL databases perform better than the SQL database we tested. We observe that even within NoSQL databases there is a wide variation in performance based on the type of operation (such as read and write). We also observe little correlation between performance and the data model each database uses.

References -

[1] G. DeCandia, et al., "Dynamo: amazon's highly available key-value
store," in SOSP '07 Proceedings of twenty-first ACM SIGOPS, New
York, USA, 2007, pp. 205-220.
[2] K. Orend, "Analysis and Classification of NoSQL Databases and
Evaluation of their Ability to Replace an Object-relational Persistence
Layer," Master Thesis, Technical University of Munich, Munich, 2010.
[3] R. Cattell, "Scalable SQL and NoSQL Data Stores," ACM SIGMOD
Record, vol. 39, December 2010.
[4] C. Strauch, "NoSQL Databases" unpublished.
[5] M. Adam, "The NoSQL Ecosystem," in The Architecture of Open
Source Applications, A. Brown and G. Wilson, Eds., lulu.com, 2011, pp.
185-204.
[6] "Project Voldemort." Internet: http://project-voldemort.com, [30.09.2011]
[7] "Redis." Internet: http://redis.io, [30.09.2011]
[8] "Membase." Internet: http://couchbase.orgjmembase, [30.09.2011]
[9] "CouchDB." Internet: http://couchdb.apache.org, [30.09.2011]
[10] "MongoDB." Internet: http://mongodb.org, [30.09.2011]
[11] "Riak." Internet: http://basho.com/Riak.html, [30.09.2011]
[12] F. Chang, et al., "Bigtable: A Distributed Storage System for Structured
Data," ACMTransactions on Computer Systems, vol. 26, pp. 1-26, 2008.

[13] "HBase." Internet: http://hbase.apache.org, [30.09.2011]
[14] "Hypertable." Internet: http://hypertable.org, [30.09.2011]
[15] "Cassandra." Internet: http://cassandra.apache.org, [30.09.2011]