NAGARJUNA COLLEGE OF ENGINEERING AND TECHOLOGY
DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Big Data

(17ISI732)

Module 5 :

HBase and Cassandra

By,

Mrs. Monika N

Asst. Professor ISE,

NCET

Module 5: HBase and Cassandra

HBase

Hadoop uses distributed file system for storing big data, and MapReduce to process it. Hadoop excels in storing and processing of huge data of various formats such as arbitrary, semi-, or even unstructured.

Limitations of Hadoop

Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.

A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

Applications such as HBase, Cassandra, couchDB, Dynamo, and MongoDB are some of the databases that store huge amounts of data and access the data in a random manner.

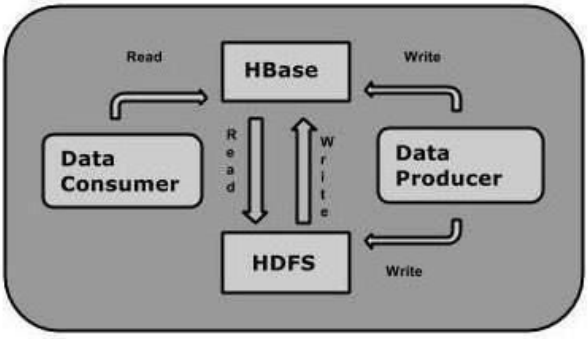
What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.
- **Namespace**: Logical grouping of tables.
- **Cell**: A {row, column, version} tuple exactly specifies a cell definition

in HBase Given below is an example schema of table in HBase.

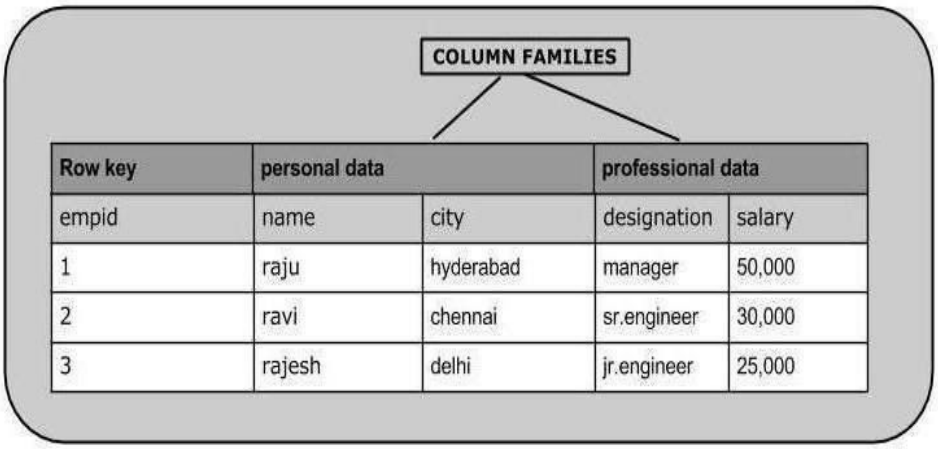
Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:



HBase and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.
Schema-less in database	Having fixed schema in database
Column-oriented databases	Row oriented data store
Wide and sparsely populated tables present in HBase	Contains thin tables in database
Supports automatic partitioning	Has no built in support for partitioning
Well suited for OLAP systems	Well suited for OLTP systems
Read only relevant data from database	Retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required
Structured and semi-structure data can be stored and processed using HBase	Structured data can be stored and processed using RDBMS
Enables aggregation over many rows and columns	Aggregation is an expensive operation

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.
- HBase is built for low latency operations
- HBase is used extensively for random read and write operations
- HBase stores a large amount of data in terms of tables
- Provides linear and modular scalability over cluster environment
- Strictly consistent to read and write operations Automatic and configurable sharding of tables
- Automatic failover supports between Region Servers
- Convenient base classes for backing Hadoop MapReduce jobs in HBase tables
- Easy to use Java API for client access
- Block cache and Bloom Filters for real-time queries Query predicate pushes down via server-side filters.

Where to Use 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 commodityhardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

Applications of HBase

- It is used whenever there is a need to write heavy applications.
- 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.

Advantages of HBase –

1. Can store large data sets
2. Database can be shared
3. Cost-effective from gigabytes to petabytes
4. High availability through failover and replication

Disadvantages of HBase – 1.

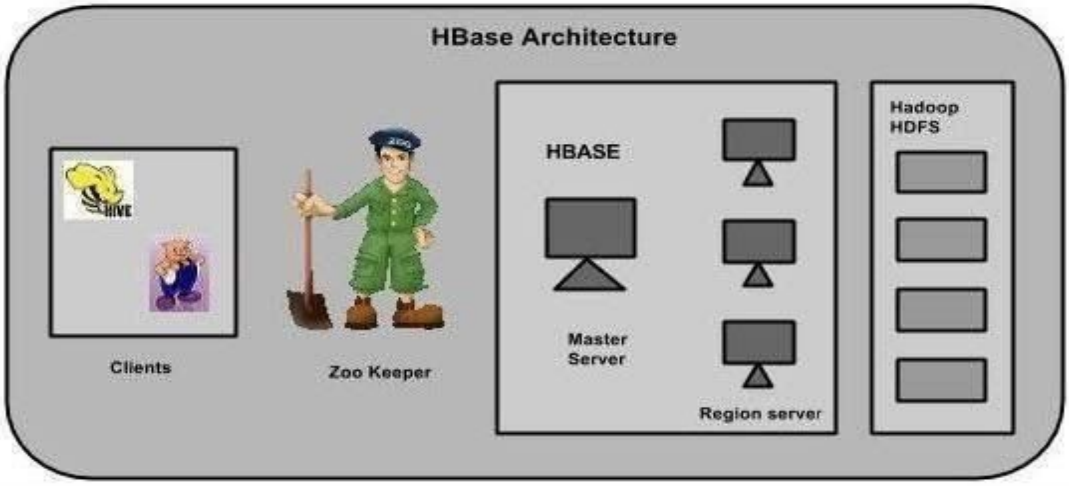
No support SQL structure

2. No transaction support
3. Sorted only on key
4. Memory issues on the cluster

HBase – Architecture

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into "Stores". Stores are saved as files in HDFS. Shown below is the architecture of HBase.

Note: The term 'store' is used for regions to explain the storage structure.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

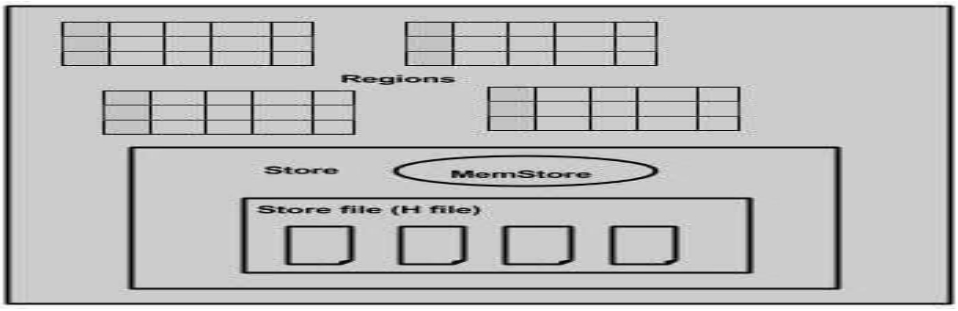
Regions are nothing but tables that are split up and spread across the region servers.

Region server

The region servers have regions that -

- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contain regions and stores as shown below:

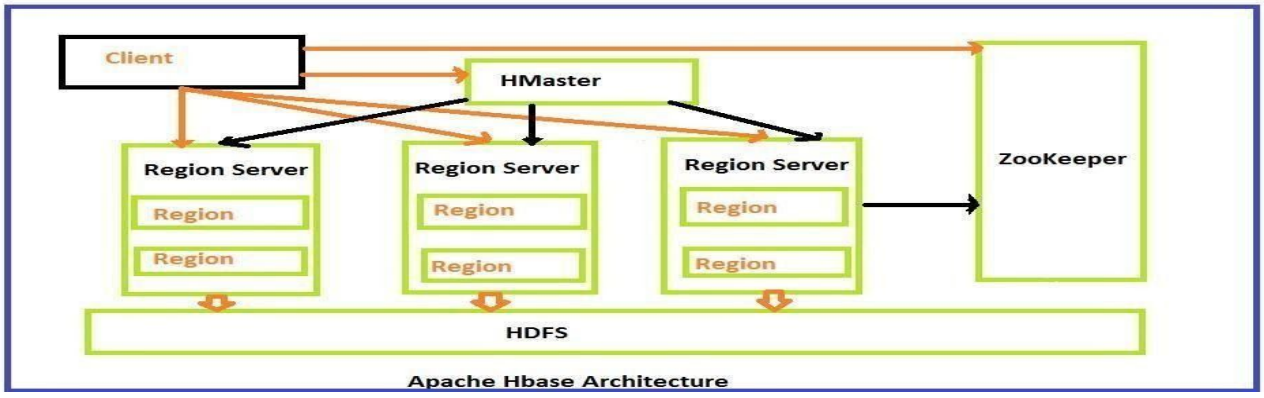


The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

HBase Architecture and its Important Components



HBase Architecture Diagram

HBase architecture consists mainly of four components

- HMaster
- HRegionserver
- HRegions
- Zookeeper
- HDFS

HMaster:

HMaster is the implementation of a Master server in HBase architecture. It acts as a monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- HMaster provides admin performance and distributes services to different region servers.
- HMaster assigns regions to region servers.
- HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
- When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

- Table (createTable, removeTable, enable, disable)
- ColumnFamily (add Column, modify Column)
- Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn, check the health status of region servers. In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

HBase Regions Servers:

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where the actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in a distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

- Hosting and managing regions
- Splitting regions automatically
- Handling read and writes requests
- Communicating with the client directly

HBase Regions:

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

ZooKeeper:

In HBase, Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the server's client has to approach ZooKeeper first.

HBase Data Model

As we know, HBase is a column-oriented NoSQL database. Although it looks similar to a relational database which contains rows and columns, but it is not a relational database. Relational databases are row oriented while HBase is column-oriented.

Row-oriented vs column-oriented Databases:

- Row-oriented databases store table records in a sequence of rows. Whereas columnoriented databases store table records in a sequence of columns, i.e. the entries in a

column are stored in contiguous locations on disks.

To better understand it, let us take an example and consider the table below.

Customer ID	Name	Address	Product ID	Product Name
1	Paul Walker	US	231	Gallardo
2	Vin Diesel	Brazil	520	Mustang

If this table is stored in a row-oriented database. It will store the records as shown below:

1, Paul Walker, US, 231, Gallardo,
2, Vin Diesel, Brazil, 520, Mustang

In row-oriented databases data is stored on the basis of rows or tuples as you can see above. While the column-oriented databases store this data as:

1,2, Paul Walker, Vin Diesel, US, Brazil, 231, 520, Gallardo, Mustang

In a column-oriented databases, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner.

- When the amount of data is very huge, like in terms of petabytes or exabytes, we use column-oriented approach, because the data of a single column is stored together and can be accessed faster.

- While row-oriented approach comparatively handles less number of rows and columns efficiently, as row-oriented database stores data is a structured format.
- When we need to process and analyze a large set of semi-structured or unstructured data, we use column oriented approach. Such as applications dealing with **Online Analytical Processing** like data mining, data warehousing, applications including analytics, etc.
- Whereas, **Online Transactional Processing** such as banking and finance domains which handle structured data and require transactional properties (ACID properties) use roworiented approach.

HBase tables has following components, shown in the image below:

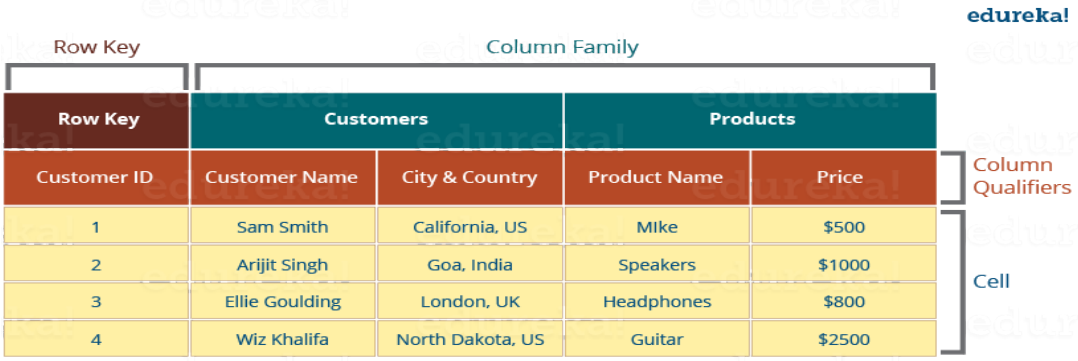


Figure: HBase Table

- **Tables**: Data is stored in a table format in HBase. But here tables are in column-oriented format.
- **Row Key**: Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
- **Column Families**: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
- **Column Qualifiers**: Each column's name is known as its column qualifier.
- **Cell**: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- **Timestamp**: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

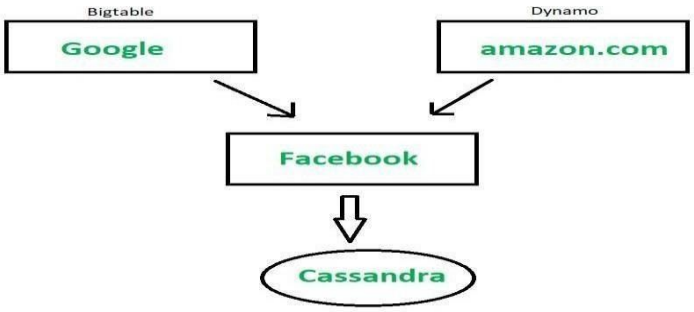
In a more simple and understanding way, we can say HBase consists of:

- Set of tables
- Each table with column families and rows Row key acts as a Primary key in HBase.
- Any access to HBase tables uses this Primary Key
- Each column qualifier present in HBase denotes attribute corresponding to the object which resides in the cell.

Cassandra

What is Apache Cassandra?

Cassandra is a distributed database management system which is open source with wide column store, NoSQL database to handle large amount of data across many commodity servers which provides high availability with no single point of failure. It is written in Java and developed by Apache Software Foundation.



Apache Cassandra is used to manage very large amounts of structure data spread out across the world. It provides highly available service with no single point of failure. Listed below are some points of Apache Cassandra:

- It is scalable, fault-tolerant, and consistent.
- It is column-oriented database.
- Its distributed design is based on Amazon's Dynamo and its data model on Google's Bigtable.
- It is Created at Facebook and it differs sharply from relational database management systems.

Cassandra implements a Dynamo-style replication model with no single point of failure but its add a more powerful "column family" data model.Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Netflix, and more.

The design goal of a Cassandra is to handle big data workloads across multiple nodes without any single point of failure.Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes of the cluster.

All the nodes of Cassandra in a cluster play the same role. Each node is independent, at the same time interconnected to other nodes. Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster. When a node goes

down, read/write request can be served from other nodes in the network.

When is Cassandra Used?

Cassandra is used when:

- A huge amount of information needs to be stored very quickly. For example, when you are processing telecom switch data or stock market data, a huge volume of data is generated every minute.
- You want the full indexed search to get the data quickly, and the data needs to be sorted in a predetermined order. Full-indexed search is search performed using a key.
- You expect an upsurge in data size. Cassandra enables scaling by adding more nodes as the data grows.
- You want a highly fault-tolerant cluster with no single point of failure. You need high performance for both data read and write.

Cassandra Program

An example of a simple Cassandra program is as shown below:

```
Create table stocks (ticker text primary key, value
int); Insert into stocks (ticker, value) values
('abc',200); Insert into stocks (ticker, value) values
('unc',400); Insert into stocks (ticker,value) values
('xyz',300); select ticker, value from stocks where
ticker = 'xyz'
;
```

```
Stocks
abc,200
xyz,300
unc,400
```

```
Result
xyz
300
```

In this example, you create a table, insert a few records, and fetch data from the table. You can see that the Cassandra syntax is very similar to the standard SQL syntax. The example creates the table called stocks with two columns, inserts three rows into this table, and selects data from the table for a particular key.Observe that Cassandra uses the primary key of the table to fetch the data. The data is also stored in the primary key order.

Advantages of Cassandra

Cassandra has many advantages for processing big data like:

- It is highly fault tolerant with no single point of failure. This means that if any node in the cluster fails, other nodes will take over and complete the work.
- Every node in the cluster is identical as there are no masters or slaves in Cassandra. Therefore, one machine cannot become the bottleneck in the system.

- Further, you can add a machine to the cluster or remove a machine from the cluster any time without downtime.
- Cassandra also provides very fast data writes allowing real-time processing of big data.
- Cassandra outperforms many other NoSQL databases regarding many performance benchmarks.

Limitations of Cassandra

Cassandra is not a general purpose database due to some following limitations:

- First, it doesn't provide aggregation of data with the group by, sum, min or max like relational databases. Any aggregation has to be pre-computed and stored.
- Second, there are no joins of tables, so data has to be denormalized before getting stored in Cassandra.

- Third, it doesn't support additional search clauses or conditions. Only keys or indexes can be used for the search. We will talk more about this restriction later in the course.
- Lastly, there is no sorting provided on non-key fields.

Features of Cassandra:

Cassandra has become popular because of its technical features. There are some of the features of cassandra:

1. **Easy data distribution –**
   It provides the flexibility to distribute data where you need by replicating data across multiple data centers.
2. **Flexible data storage –**
   Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured.It can dynamically accommodate changes to your data structures accordingly to your need.
3. **Elastic scalability –**
   Cassandra is highly scalable and allows to add more hardware to accommodate more customers and more data as per requirement.
4. **Fast writes –**
   Cassandra was designed to run on cheap commodity hardware.Cassandra performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.
5. **Always on Architecture –**
   Cassandra has no single point of failure and it is continuously available for business-critical applications that connot afford a failure.
6. **Fast linear-scale performance –**
   Cassandra is linearly scalable therefore it increases your throughput as you increase the number of nodes in the cluster.It maintains a quick response time.
7. **Transaction support –**
   Cassandra supports properties like Atomicity, Consistency, Isolation, and Durablity (ACID) properties of transactions.

Layers of a Cassandra Cluster

- **Cluster** – Cassandra Cluster is a collection of nodes in a ring format that work together. It can span multiple physical locations. Data is distributed across nodes in the cluster using consistent hashing-based function.
- **Node** -Node is the basic infrastructure component of Cassandra and it's where the data is stored. Each node contains a data replica.
- **Keyspace** – Keyspace is a collection of column families and is equivalent to a database in RDBMS. The definition of keyspace contains Replication factor, Replication strategy (simple or network topology) and Column families.

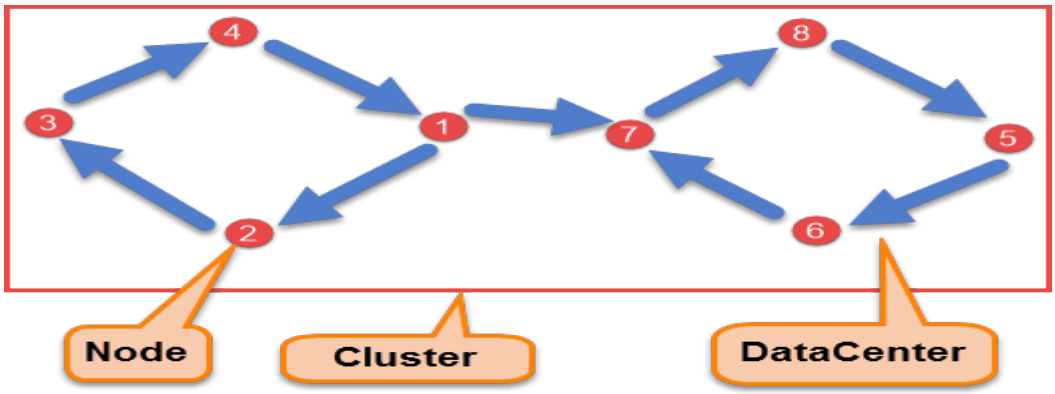
- **Column Family** – Column family is a container of a collection of rows. This is equivalent to a table in RDBMS.
- ***Row*** – Each row in Cassandra is identified by a unique key and each row can have different columns. The row key is a unique string and there is no limit on its size.
- ***Column***– Each Column is a construct that has a name, a value and a user-defined timestamp with it. Each row can have a variable number of columns.

Cassandra Applications

1. Cassandra Storage
2. Back-end development applications
3. Cassandra Monitoring
4. Time-series-based applications
5. Cassandra Analytics
6. Cassandra Messaging

Architecture of Cassandra

Some of the key components of the Cassandra architecture are as follows:



- **Cluster:** It is a complete set of multiple data centers on which the entire data is stored for processing in the Cassandra NoSQL database.
- **Data center:** A set of related nodes are grouped in a data center.
- **Node:** The specific place where the data resides on the cluster is called a node.
- **Commit log:** It is a failsafe method that is deployed by Cassandra in order to take a backup of all data in the Cassandra database by writing it to the commit log.
- **Memtable:** It is a data structure that resides in the memory where Cassandra buffers writes. There will be one active Memtable per table.
- **SSTable:** When Memtables reach their threshold value, they are flushed onto the disk, and they become immutable SSTables.

- **Bloom filter:** The bloom filter is an algorithm that lets you test whether an element is a member of a set in a swift manner. These bloom filters are accessed after each query.

Data Replication in Cassandra

As hardware problem can occur or link can be down at any time during data process, a solution is required to provide a backup when the problem has occurred. So data is replicated for assuring no single point of failure.

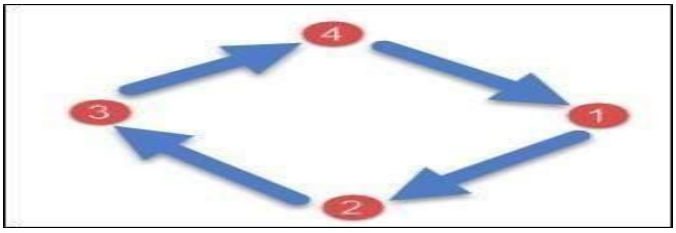
Cassandra places replicas of data on different nodes based on these two factors.

- Where to place next replica is determined by the **Replication Strategy**.
- While the total number of replicas placed on different nodes is determined by the **Replication Factor**.

One Replication factor means that there is only a single copy of data while three replication factor means that there are three copies of the data on three different nodes.For ensuring there is no single point of failure, **replication factor must be three.** There are two kinds of replication strategies in Cassandra.

Simple Strategy

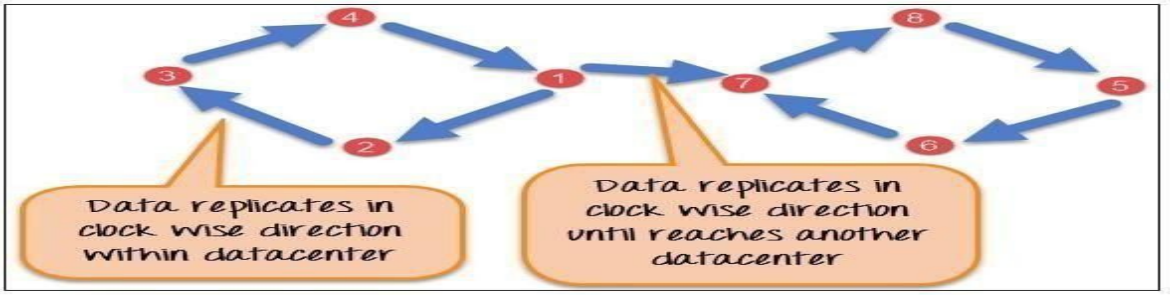
SimpleStrategy is used when you have just one data center. SimpleStrategy places the first replica on the node selected by the partitioner. After that, remaining replicas are placed in clockwise direction in the Node ring. Here is the pictorial representation of the SimpleStrategy.



Network Topology Strategy

NetworkTopologyStrategy is used when you have more than two data centers. In NetworkTopologyStrategy, replicas are set for each data center separately.

NetworkTopologyStrategy places replicas in the clockwise direction in the ring until reaches the first node in another rack. This strategy tries to place replicas on different racks in the same data center. This is due to the reason that sometimes failure or problem can occur in the rack. Then replicas on other nodes can provide data. Here is the pictorial representation of the Network topology strategy



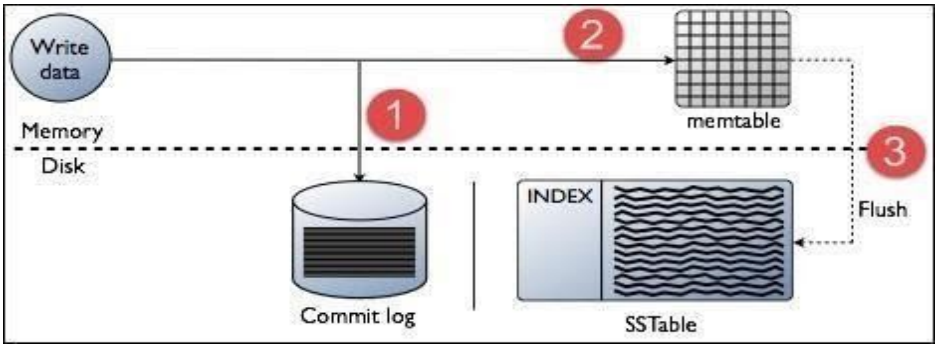
Write Operation

The coordinator sends a write request to replicas. If all the replicas are up, they will receive write request regardless of their consistency level. **Consistency level** determines how many nodes will respond back with the success acknowledgment. The node will respond back with the success acknowledgment if data is written successfully to the commit log and **memTable.**

For example, in a single data center with replication factor equals to three, three replicas will receive write request. If consistency level is one, only one replica will respond back with the success acknowledgment, and the remaining two will remain dormant. Suppose if remaining two replicas lose data due to node downs or some other problem, Cassandra will make the row consistent by the built-in repair mechanism in Cassandra.

Here it is explained, how write process occurs in Cassandra,

1. When write request comes to the node, first of all, it logs in the commit log.
2. Then Cassandra writes the data in the mem-table. Data written in the mem-table on each write request also writes in commit log separately. Mem-table is a temporarily stored data in the memory while Commit log logs the transaction records for back up purposes.
3. When mem-table is full, data is flushed to the SSTable data file.



Read Operation

There are three types of read requests that a coordinator sends to replicas.

1. Direct request
2. Digest request
3. Read repair request

The coordinator sends direct request to one of the replicas. After that, the coordinator sends the digest request to the number of replicas specified by the consistency level and checks whether the returned data is an updated data.

After that, the coordinator sends digest request to all the remaining replicas. If any node gives out of date value, a background read repair request will update that data. This process is called read repair mechanism.

Cassandra - Data Model

The data model of Cassandra is significantly different from what we normally see in an RDBMS. This chapter provides an overview of how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

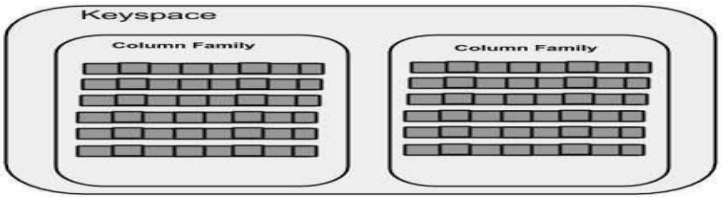
Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are −

- **Replication factor** − It is the number of machines in the cluster that will receive copies of the same data.
- **Replica placement strategy** − It is nothing but the strategy to place replicas in the ring. We have strategies such as **simple strategy** (rack-aware strategy), **old network topology strategy** (rack-aware strategy), and **network topology strategy** (datacentershared strategy).
- **Column families** − Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows −

CREATE KEYSPACE Keyspace name

WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3}; The following illustration shows a schematic view of a Keyspace.



Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

Relational Table	Cassandra column Family
A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value.	In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time.
Relational tables define only columns and the user fills in the table with values.	In Cassandra, a table contains columns, or can be defined as a super column family.

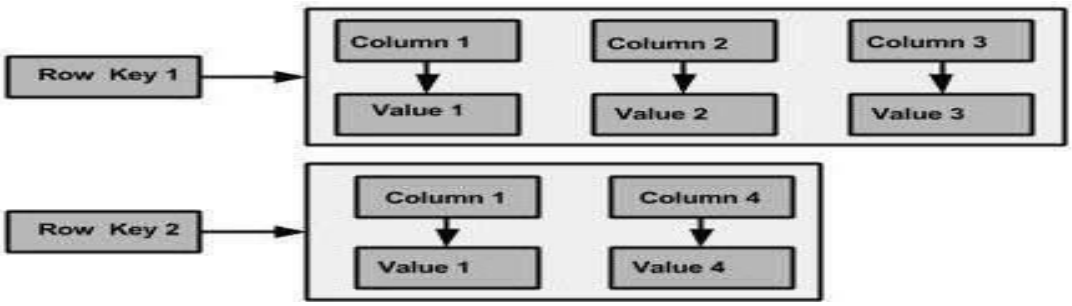
A Cassandra column family has the following attributes −

- **keys_cached** − It represents the number of locations to keep cached per SSTable.
- **rows_cached** − It represents the number of rows whose entire contents will be cached in memory.
- **preload_row_cache** − It specifies whether you want to pre-populate the row cache.

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. (ROW x COLUMN)	In Cassandra, a table is a list of "nested key-value pairs". (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Note − Unlike relational tables where a column family's schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.



Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Column		
name : byte[]	value : byte[]	clock : clock[]

SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same

column family, and a super column can be helpful here.Given below is the structure of a super column.

Super Column	
name : byte[]	cols : map<byte[], column>

Cassandra Query Language(CQL): Insert Into, Update, Delete (Example) Insert Data

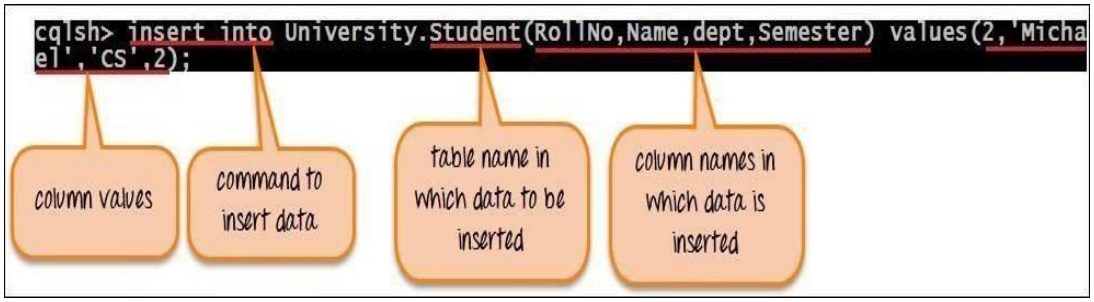
Command 'Insert into' writes data in Cassandra columns in row form. It will store only those columns that are given by the user. You have to necessarily specify just the primary key column. It will not take any space for not given values. No results are returned after insertion.

Syntax

```
Insert into KeyspaceName.TableName(ColumnName1, ColumnName2, ColumnName3 ) values
(Column1Value, Column2Value, Column3Value )
```

Example

Here is the snapshot of the executed command 'Insert into' that will insert one record in Cassandra table 'Student'.



Insert into University.Student(RollNo,Name,dept,Semester) values(2,'Michael','CS', 2);

After successful execution of the command 'Insert Into', one row will be inserted in the Cassandra table Student with RollNo 2, Name Michael, dept CS and Semester 2. Here is the

snapshot of the current database state.

```
rollno | dept | name    | semester
--------+------+---------+----------
      2 |   CS | Michael |        2
(1 rows)
```

Upsert Data

Cassandra does upsert. Upsert means that Cassandra will insert a row if a primary key does not exist already otherwise if primary key already exists, it will update that row.

Update Data

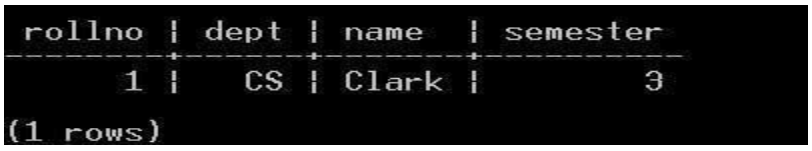
Command 'Update' is used to update the data in the Cassandra table. If no results are returned after updating data, it means data is successfully updated otherwise an error will be returned. Column values are changed in 'Set' clause while data is filtered with 'Where' clause.

Syntax

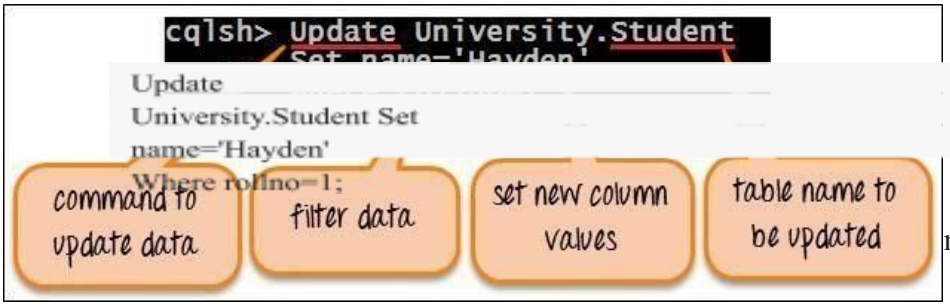
```
Update KeyspaceName.TableName
Set                ColumnName1=new
    Column1Value,
    ColumnName2=new
    Column2Value,
    ColumnName3=new
    Column3Value,
      .
      .
      .
Where ColumnName=ColumnValue
```

Example

Here is the screenshot that shows the database state before updating data.



Here is the snapshot of the executed command 'Update' that update the record in the Student table.



name will be changed

Here is the screenshot that shows the database state after updating data.

Cassandra Delete Data

Command 'Delete' removes an entire row or some columns from the table Student. When data is deleted, it is not deleted from the table immediately. Instead deleted data is marked with a tombstone and are removed after compaction.

Syntax

```
Delete from KeyspaceName.TableName
        Where
        ColumnName1=ColumnValue
```

The above syntax will delete one or more rows depend upon data filtration in where clause.

```
Delete ColumnNames from KeyspaceName.TableName
    Where ColumnName1=ColumnValue
```

The above syntax will delete some columns from the table.

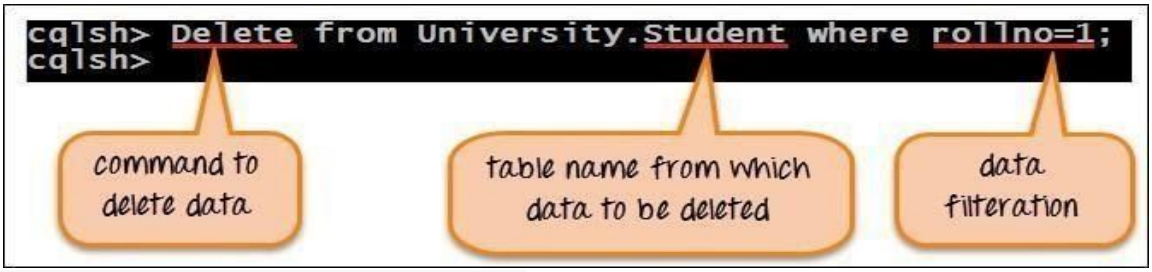
Example

Here is the snapshot that shows the current database state before deleting data.

```
 rollno | dept | name    | semester
--------+------+---------+----------
      1 |   CS | Hayden  |        3
      2 |   CS | Guru99  |        7

(2 rows)
```

Here is the snapshot of the command that will remove one row from the table Student.



Delete from University.Student where rollno=1;

After successful execution of the command 'Delete', one rows will be deleted from the table Student where rollno value is 1.

Here is the snapshot that shows the database state after deleting data.

```
 rollno | dept | name    | semester
--------+------+---------+----------
      2 |   CS | Guru99  |        7

(1 rows)
```

What Cassandra does not support

There are following limitations in Cassandra query language (CQL).

1. CQL does not support aggregation queries like max, min, avg 2. CQL does not support group by, having queries.
3. CQL does not support joins.
4. CQL does not support OR queries.
5. CQL does not support wildcard queries.
6. CQL does not support Union, Intersection queries.
7. Table columns cannot be filtered without creating the index.
8. Greater than (>) and less than (<) query is only supported on clustering column.

Cassandra query language is not suitable for analytics purposes because it has so many limitations.

Cassandra Where Clause

In Cassandra, data retrieval is a sensitive issue. The column is filtered in Cassandra by creating an index on non-primary key columns.

Syntax

```
Select ColumnNames from KeyspaceName.TableName Where
ColumnName1=Column1Value AND
        ColumnName2=Column2Value AND
        .
        .
        .
```

Example

- Here is the snapshot that shows the data retrieval from Student table without data filtration.

```
cqlsh> select * from University.Student;

 rollno | dept | name   | semester
--------+------+--------+----------
      1 |   CS | Jeegar |        5
      2 |   CS | Guru99 |        7

(2 rows)
cqlsh>
```

sity.Student;

Two records are retrieved from Student table.

Here is the snapshot that shows the data retrieval from Student with data filtration. One record is retrieved.

Data is filtered by name column. All the records are retrieved that has name equal to Guru99.

```
cqlsh> select * from University.Student where name='Guru99';

 rollno | dept | name   | semester
--------+------+--------+----------
      2 |   CS | Guru99 |        7

(1 rows)
cqlsh>
```

```
select * from University.Student where
name='Guru99';
```

Below are common operation we can do with CQL **1. Creating and using key space:**

```
cqlsh> CREATE KEYSPACE demodb WITH REPLICATION = { 'class'
: 'NetworkTopologyStrategy', 'datacenter1' : 3 };
```

USE demodb;

2. Alter Key Space

ALTER KEYSPACE ” demodb ” WITH REPLICATION = { ‘class’ : ‘SimpleStrategy’, ‘replication_factor’ : 2};

3. Create Table

CREATE TABLE emp (empID int, deptID int, first_name varchar,last_name varchar, PRIMARY KEY (empID, deptID));

4. Insert into table

INSERT INTO emp (empID, deptID, first_name, last_name) VALUES (104, 15, ‘jane’, ‘smith’);

5. Select query Select * from emp;