Apache Cassandra Overview

What is Cassandra?

Apache Cassandra is highly scalable, high performance, distributed NoSQL database. Cassandra is designed to handle huge amount of data across many commodity servers, providing high availability without a single point of failure.

Also, we can say, Apache Cassandra is a **distributed**, **decentralized**, **highly available**, **scalable, large and growing data**, **high availability**, **changing data**, **eventual consistency and faster write NoSQL database**. ***Distributed*** means it is capable of running on multiple machines sharing load on different machines ***Decentralized*** means there is no master-slave concept. All nodes are identical and there is no Single Point of Failure. Whenever one node goes down then other node takes over the charge. Thus, Cassandra is a ***highly available*** database. ***Scalable*** means at any given point of time we can add or remove any node with very little performance impact on performance. There is no manual intervention no manual rebalancing. Everything happens automatically in the background.

Cassandra is useful where there is ***large and growing data*** and ***high availability*** is important. Cassandra is useful where ***data is changing***. Cassandra may not be a good option where data is not changing and static. It is useful where consistency is not a concern. Cassandra is ***eventual consistent*** database by default. That means if you write something and read that one immediately, then it may be found that the data is inconsistent. So, it may cause some performance impact. If our application demands consistency, then Cassandra may not be a good option. Cassandra has a very high throughput for ***faster write***.

Cassandra supports ‘Active everywhere design’ means all nodes can be written to and read from.

Cassandra is a NoSQL database

NoSQL database is non-relational database. It is also called Not Only SQL. It is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

Reasons behind its popularity

Cassandra is an Apache product. It is an open source, distributed and decentralized/distributed storage system (database). It is used to manage very large amounts of structured data spread out across the world. It provides high availability with no single point of failure.

Important Points of Cassandra

- Cassandra is a column-oriented database.
- Cassandra is scalable, consistent, and fault tolerant.
- Cassandra's distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.
- Cassandra is created at Facebook. It is totally different from relational database management systems.
- Cassandra follows a Dynamo-style replication model with no single point of failure but adds a more powerful "column family" data model.
- Cassandra is being used by some of the biggest companies like Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

History of Cassandra

Cassandra was initially developed at Facebook by two Indians Avinash Lakshman (one of the authors of Amazon's Dynamo) and Prashant Malik. It was developed to power the Facebook inbox search feature.

The following points specify the most important happenings in Cassandra history:

- Cassandra was developed at Facebook by Avinash Lakshman and Prashant Malik.
- It was developed for Facebook inbox search feature.
- It was open sourced by Facebook in July 2008.
- It was accepted by Apache Incubator in March 2009.
- Cassandra is a top-level project of Apache since February 2010.
- The latest version of Apache Cassandra is 4.3.0 released on 2022-02-17 and will be maintained until release (May-July 2024).

Features of Cassandra

There are a lot of outstanding technical features which makes Cassandra very popular. Following is a list of some popular features of Cassandra:

- **High Scalability -** Cassandra is highly scalable which facilitates you to add more hardware to attach more customers and more data as per requirement.
- **Rigid Architecture -** Cassandra has not a single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- **Fast Linear-scale Performance -** Cassandra is linearly scalable. It increases your throughput because it facilitates you to increase the number of nodes in the cluster. Therefore, it maintains a quick response time.
- **Fault tolerant -** Cassandra is fault tolerant. Suppose there are 4 nodes in a cluster, here each node has a copy of same data. If one node is no longer serving, then other three nodes can serve as per request.
- **Flexible Data Storage -** Cassandra supports all possible data formats like structured, semi-structured, and unstructured. It facilitates you to make changes to your data structures according to your need.
- **Easy Data Distribution -** Data distribution in Cassandra is very easy because it provides the flexibility to distribute data where you need by replicating data across multiple data centres.
- **Transaction Support -** Cassandra supports properties like **A**tomicity, **C**onsistency, **I**solation, and **D**urability (ACID).
- **Fast writes -** Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

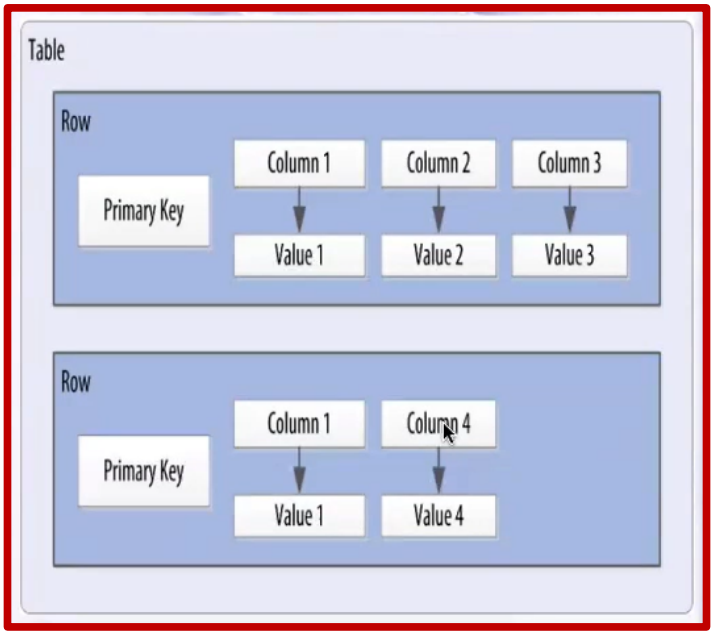
Cassandra Architecture

Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Cassandra Data Model

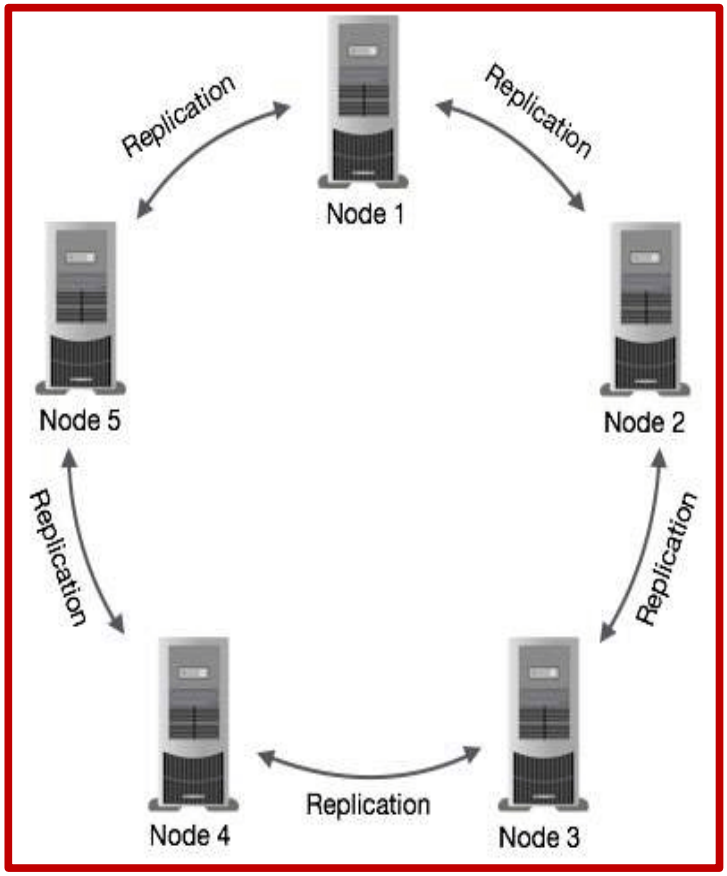
In Cassandra ***Column*** is the most basic unit and it is nothing but name value-pair. ***Row*** is a container for columns referenced by primary key. If a column has the null value, then we don't need to keep that column. As in this diagram in row-1 we have column 1, 2 and 3. But in row -2 we have only column 1 and 2 only. This type of architecture saves lots of space. ***Table*** is a container of rows. So inside table we have rows and inside rows we have columns. ***Keyspace*** is a container for tables. In RDBMS we have database concept, here in Cassandra we have keyspace concept. Keyspace does not reside on a single machine. It spans over multiple machines. And lastly ***Cluster*** is a container for keyspace. A cluster may contain multiple data centres. So, this is known as Cassandra data model.



Data Replication in Cassandra

In Cassandra, nodes in a cluster act as replicas for a given piece of data. If some of the nodes are responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.

See the following image to understand the schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.



Note: A Single Point of Failure (SPOF) is **any non-redundant part of a system that, if dysfunctional, would cause the entire system to fail**. A single point of failure is a hurdle to the goal of high availability in a computing system or network, a software application, a business practice, or any other industrial system.

Components of Cassandra

The main components of Cassandra are

- **Node:** A Cassandra node is a place where data is stored.
- **Data centre:** Data centre is a collection of related nodes.
- **Cluster:** A cluster is a component which contains one or more data centres.
- **Commit log:** In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.
- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Query Language

Cassandra Query Language (CQL) is used to access Cassandra through its nodes. CQL treats the database (Keyspace) as a container of tables. Programmers use cqlsh: a prompt to work with CQL or separate application language drivers.

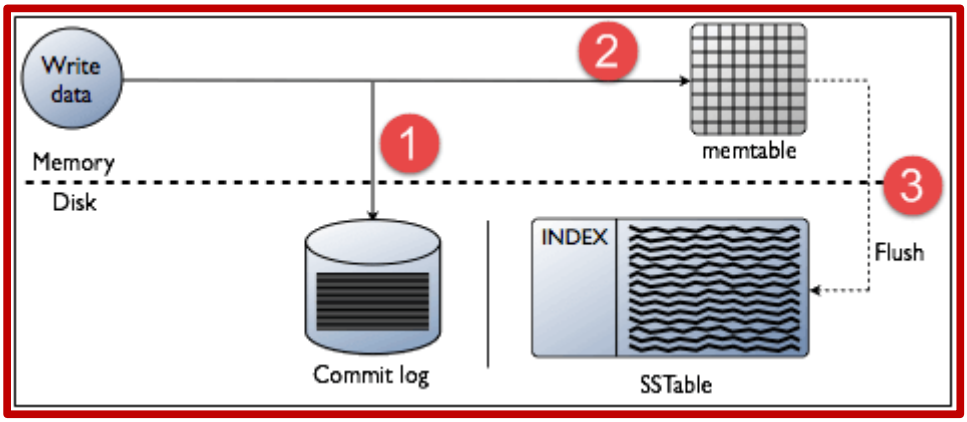
The client can approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

Cassandra Constraints

Cassandra supports ***no join***. We cannot join two tables. There is ***no foreign key*** concept. We cannot refer one attribute of one table with another attribute of another table. These operations are very expensive in distributed database. Cassandra ***does not support flexible queries***. That means that we cannot use any column in the where clause. We should understand our application at first and what are the queries we need in advance.

Write Operations

Every write activity of nodes is captured by the commit logs written in the nodes. Later the data will be captured and stored in the mem-table. Whenever the mem-table is full, data will be written into the SStable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.



Read Operations

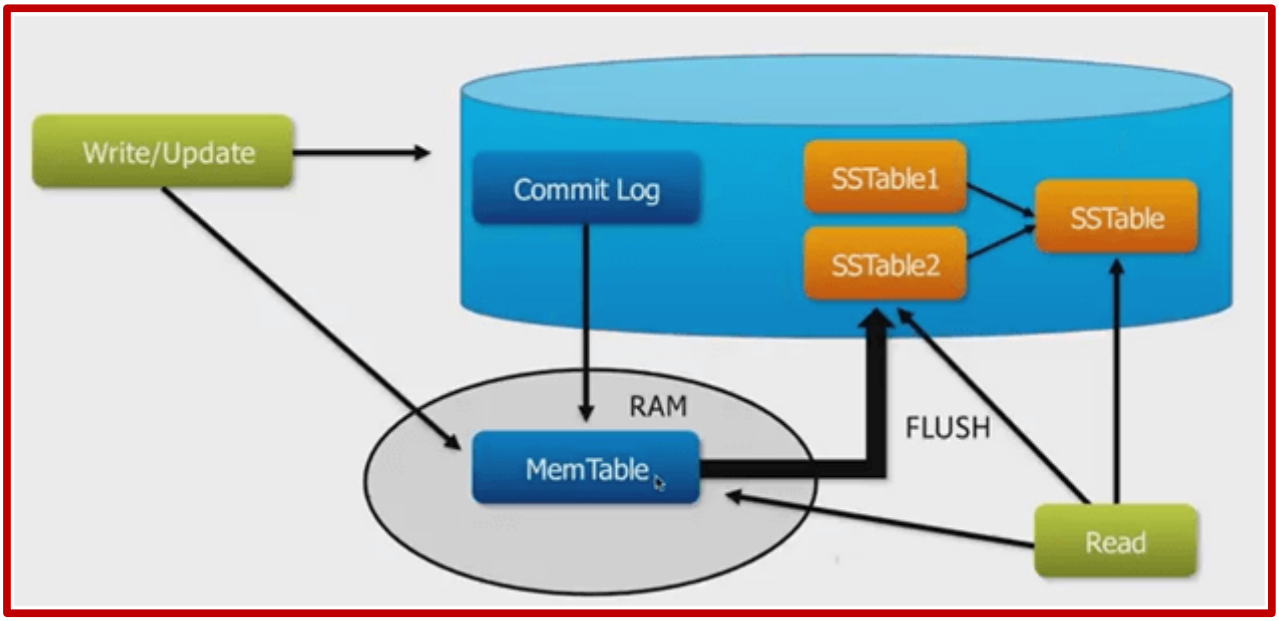
In Read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable which contains the required data.

There are three types of read request that is sent to replicas by coordinators.

- Direct request
- Digest request
- 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 if the returned data is an updated data.

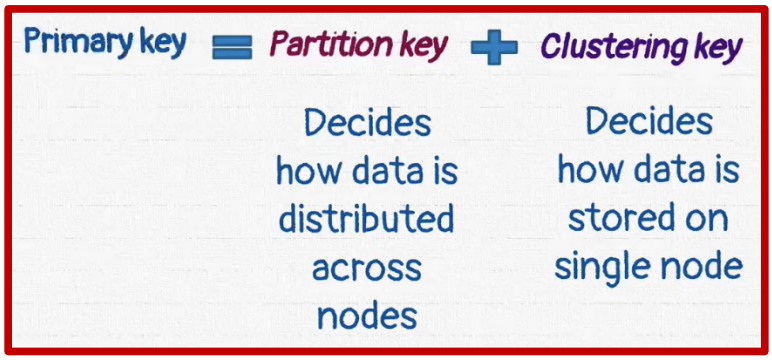
After that, the coordinator sends digest request to all the remaining replicas. If any node gives out of date value, a background read repair request will update that data. This process is called read repair mechanism.

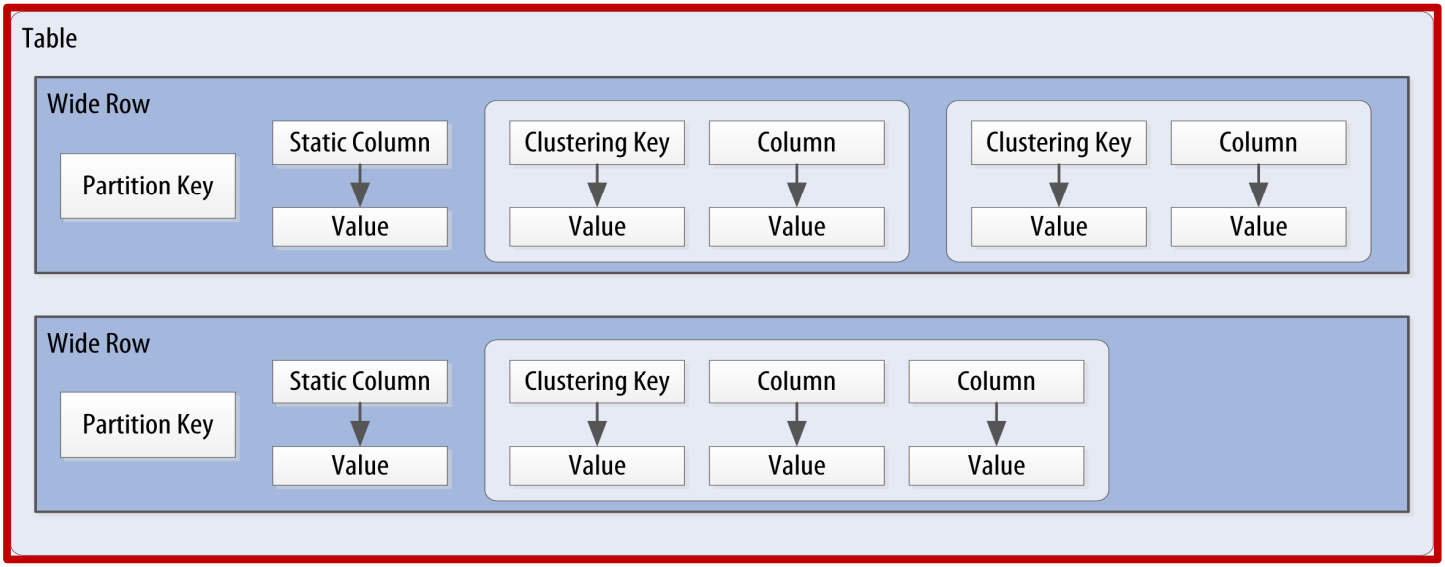


The simplest and the fastest read path is when the record is present in the Row Cache.

Cassandra Keys

In Cassandra, ***Primary Key*** concept differs from RDBMS Primary Key concept. In Cassandra Primary Key has two parts – ***Partition Key*** (Which decides how data is distributed across nodes) and ***Clustering Key*** (Which decides how data is stored on single node). The below image can give us better idea about the implementation of a wide row in Cassandra.





How is data distributed in Cassandra?

Every node is assigned a unique token or a range of tokens that determines which row will go to which node. So, every node is assigned with a finite range of tokens. Now how these tokens are generated? There is a ***partitioner*** class. Which is having a hashing function called ***partitioner***. That is used to generate these token values.

Let us take an example. Actually, in practice these tokens are generated using some complex algorithms. Now let us consider we are having -

- **Token range = 1 to 100**
- **Number of nodes = 5**
- **Token range per node = 100 / 5 = 20**

And these tokens are distributed as shown in the diagram. Let say we want to write a row with a partition key against the emp_id = 20. And the hashing function partitioner on emp_id generates the partition key 62.

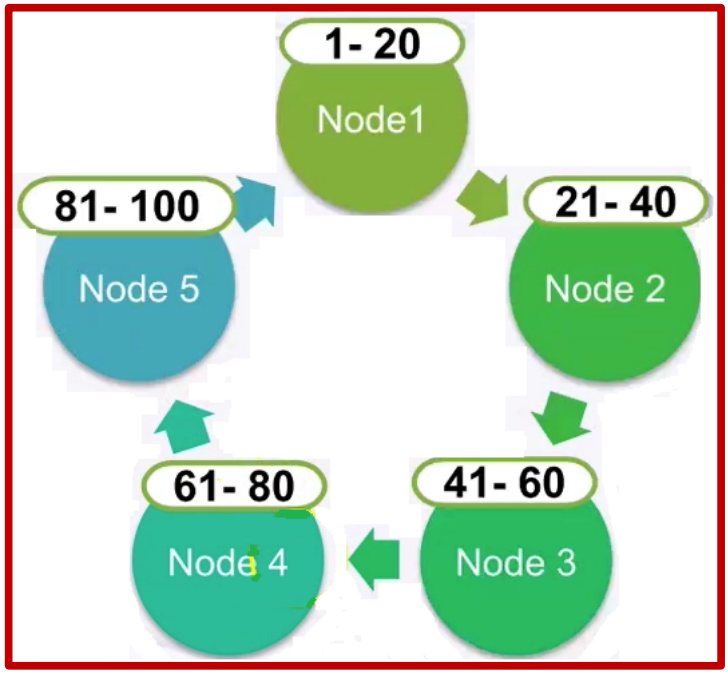
Partitioner(emp_id) -> 62

So, this 62 means the row will be mapped with the node number 4 as it is having the token range from 61 to 80.

So always for the emp_id = 20 will generate the partition key 62 for all read and write operation requests.

Now let us discuss how clustering key determines how data is stored with in a single machine. Suppose that we have a row with primary key pk1 and followed by clustering key ck1-a and ck1-b and that followed by rest of the columns.

When against pk1 there are multiple rows then rows are stored as sorted by the values of clustering keys.



pk1	ck1-a	ck1-b	Rest of columns

pk1	ck1-a	ck1-b	Rest of columns
	ck2-a	ck2-b	Rest of columns
	ck3-a	ck3-b	Rest of columns
	ck4-a	ck4-b	Rest of columns

Relational database vs. NoSQL database

Cassandra is a NoSQL database. The main objective of a NoSQL database is to have the following three things:

- Simplicity of design
- Horizontal scaling
- High availability

NoSQL is faster than relational database management system because it uses different data structure compared to relational databases.

Cassandra data structure is faster than relational database structure. NoSQL databases are mainly used in Bigdata and real time web applications.

Differences between NoSQL and Relational database

NoSQL Database	Relational Database
NoSQL Database supports a very simple query language.	Relational Database supports a powerful query language.
NoSQL Database has no fixed schema.	Relational Database has a fixed schema.
NoSQL Database is only eventually consistent.	Relational Database follows ACID properties. (**A**tomicity, **C**onsistency, **I**solation, and **D**urability)
NoSQL databases don't support transactions (support only simple transactions).	Relational Database supports transactions (also complex transactions with joins).
NoSQL Database is used to handle data coming in high velocity.	Relational Database is used to handle data coming in low velocity.
The NoSQL's data arrive from many locations.	Data in relational database arrive from one or few locations.
NoSQL database can manage structured, unstructured, and semi-structured data.	Relational database manages only structured data.
NoSQL databases have no single point of failure.	Relational databases have a single point of failure with failover.
NoSQL databases can handle big data or data in a very high volume.	NoSQL databases are used to handle moderate volume of data.
NoSQL has decentralized structure.	Relational database has centralized structure.
NoSQL database gives both read and write scalability.	Relational database gives read scalability only.
NoSQL database is deployed in horizontal fashion.	Relation database is deployed in vertical fashion.

Use Cases / Applications of Cassandra

Cassandra can be used for different type of applications. Following is a list of use cases where Cassandra should be preferred:

- **Messaging:** Cassandra is a great database which can handle a big amount of data. So, it is preferred for the companies that provide Mobile phones and messaging services. These companies have a huge amount of data, so Cassandra is best for them.
- **Handle high speed Applications:** Cassandra can handle the high-speed data, so it is a great database for the applications where data is coming at very high speed from different devices or sensors.
- **Product Catalog and retail apps:** Cassandra is used by many retailers for durable shopping cart protection and fast product catalog input and output.
- **Social Media Analytics and recommendation engine:** Cassandra is a great database for many online companies and social media providers for analysis and recommendation to their customers.

Cassandra Data Types

Cassandra supports different types of data types. They can be classified in multiple groups as shown below:

Numeric data type: int, bigint, smallint, tinyint, varint, float, double, decimal

Textual data type: text, varchar

Collection data type: set, list, map

Other data type: Boolean, blob (binary large object), uuid (universally unique identifier), timeuuid, user-defined

Let's see the different data types in the following table:

CQL Type	Constants	Description
ascii	Strings	US-ascii character string
bigint	Integers	64-bit signed long
blob	Blobs (binary large object)	Arbitrary bytes in hexadecimal
boolean	Booleans	True or False
counter	Integers	Distributed counter values 64 bit
decimal	Integers, Floats	Variable precision decimal
double	Integers, Floats	64-bit floating point
float	Integers, Floats	32-bit floating point
frozen	Tuples, collections, user defined types	stores Cassandra types
inet	Strings	IP address in ipv4 or ipv6 format
int	Integers	32-bit signed integer
list		Collection of elements
map		JSON style collection of elements
set		Collection of elements
text	strings	UTF-8 encoded strings
timestamp	Integers, Strings	ID generated with date plus time
uuid	Uuids ("**UUID**" stands for "universally unique identifier").	It is 128-bit Hex value and supposed to be unique even beyond the table. Convenience function uuid().
timeuuid	Unique identifier. Based on MAC address, system time and a sequence number. Prevent duplicates.	Convenience functions are now(), dateOf(), unixTimestampOf().
varchar	strings	UTF-8 encoded string
varint	Integers	Arbitrary precision integer
tuple		A group of 2,3 fields

Cassandra Automatic Data Expiration

Cassandra provides functionality by which data can be automatically expired.

During data insertion, you must specify 'ttl' value in seconds. 'ttl' value is the time to live value for the data. After that amount of time, data will be automatically removed.

Cassandra Data Model

Data model in Cassandra is totally different from normally we see in RDBMS. Let's see how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that are operated together. The outermost container is known as the Cluster which contains different nodes. 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. Following are the basic attributes of Keyspace in Cassandra:

- **Replication factor:** It specifies the number of machines in the cluster that will receive copies of the same data.
- **Replica placement Strategy:** It is a strategy which species how to place replicas in the ring. There are three types of strategies such as:
    1) Simple strategy (rack-aware strategy)
    2) old network topology strategy (rack-aware strategy)
    3) network topology strategy (datacentre-shared strategy)
- **Column families:** column families are placed under keyspace. A keyspace is a container for a list of one or more column families while a column family 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.

In Cassandra, a well data model is very important because a bad data model can degrade performance, especially when you try to implement the RDBMS concepts on Cassandra.

Cassandra data Models Rules

- Cassandra doesn't support JOINS, GROUP BY, OR clause, aggregation etc. So, you must store data in a way that it should be retrieved whenever you want.
- Cassandra is optimized for high write performances so you should maximize your writes for better read performance and data availability. There is a trade-off between data write and data read. So, optimize your data read performance by maximizing the number of data writes.
- Maximize data duplication because Cassandra is a distributed database and data duplication provides instant availability without a single point of failure.

Data Modelling Goals

You should have following goals while modelling data in Cassandra:

- **Spread Data Evenly Around the Cluster:** To spread equal amount of data on each node of Cassandra cluster, you must choose integers as a primary key. Data is spread to different nodes based on partition keys that are the first part of the primary key.
- **Minimize number of partitions read while querying data:** Partition is used to bind a group of records with the same partition key. When the read query is issued, it collects data from different nodes from different partitions.

In the case of many partitions, all these partitions need to be visited for collecting the query data. It does not mean that partitions should not be created. If your data is very large, you can't keep that huge amount of data on the single partition. The single partition will be slowed down. So, you must have a balanced number of partitions.

Cassandra Vs HBase

The following table specifying the main differences between Cassandra and HBase:

HBase	Cassandra
HBase is based on Bigtable (Google)	Cassandra is based on DynamoDB (Amazon). It was initially developed at Facebook by former Amazon engineers. This is one reason why Cassandra supports multi data centre.
HBase uses the Hadoop infrastructure (Zookeeper, NameNode, HDFS). Organizations that deploy Hadoop must have the knowledge of Hadoop and HBase	Cassandra started and evolved separate from Hadoop and its infrastructure and operational knowledge requirements are different than Hadoop. However, for analytics, many Cassandra deployments use Cassandra + Storm (which uses zookeeper), and/or Cassandra + Hadoop.
The HBase-Hadoop infrastructure has several "moving parts" consisting of Zookeeper, Name Node, HBase master, and data nodes, Zookeeper is clustered and naturally fault tolerant. Name Node needs to be clustered to be fault tolerant.	Cassandra uses a single node-type. All nodes are equal and perform all functions. Any node can act as a coordinator, ensuring no Single Point of Failure (SPOF). Adding storm or Hadoop, of course, adds complexity to the infrastructure.
HBase is well suited for doing range-based scans.	Cassandra does not support range-based row-scans which may be limiting in certain use-cases.
HBase provides for asynchronous replication of an HBase cluster across a wan.	Cassandra random partitioning provides for row-replication of a single row across a wan.
HBase only supports ordered partitioning.	Cassandra officially supports ordered partitioning, but no production user of Cassandra uses ordered partitioning due to the "hot spots" it creates and the operational difficulties such hot-spots cause.
Due to ordered partitioning, HBase will easily scale horizontally while still supporting Rowkey range scans.	If data is stored in columns in Cassandra to support range scans, the practical limitation of a row size in Cassandra is 10's of megabytes.
HBase supports atomic compare and set. HBase supports transaction within a row.	Cassandra does not support atomic compare and set.
HBase does not support read load balancing against a single row. A single row is served by exactly one region server at a time.	Cassandra will support read load balancing against a single row.
Bloom filters can be used in HBase as another form of indexing.	Cassandra uses bloom filters for key lookup.
Triggers are supported by the coprocessor capability in HBase.	Cassandra does not support co-processor-like functionality.

Cassandra Vs RDBMS

The following table specifying the main differences between Cassandra and RDBMS:

Cassandra	RDBMS
Cassandra is used to deal with unstructured data.	RDBMS is used to deal with structured data.
Cassandra has flexible schema.	RDBMS has fixed schema.
In Cassandra, a table is a list of "nested key-value pairs". (Row x Column Key x Column value)	In RDBMS, a table is an array of arrays. (Row x Column)
In Cassandra, keyspace is the outermost container which contains data corresponding to an application.	In RDBMS, database is the outermost container which contains data corresponding to an application.
In Cassandra, tables or column families are the entity of a keyspace.	In RDBMS, tables are the entities of a database.
In Cassandra, row is a unit of replication.	In RDBMS, row is an individual record.
In Cassandra, column is a unit of storage.	In RDBMS, column represents the attributes of a relation.
In Cassandra, relationships are represented using collections.	In RDBMS, there are concept of foreign keys, joins etc.

Cassandra CQLsh

Cqlsh provides a lot of options which you can see in the following table:

Options	Usage
help	This command is used to show help topics about the options of Cqlsh commands.
version	it is used to see the version of the Cqlsh you are using.
color	it is used for coloured output.
debug	It shows additional debugging information.
execute	It is used to direct the shell to accept and execute a CQL command.
file= "file name"	By using this option, Cassandra executes the command in the given file and exits.
no-color	It directs Cassandra not to use coloured output.
u "username"	Using this option, you can authenticate a user. The default username is: Cassandra.
p "password"	Using this option, you can authenticate a user with a password. The default password is Cassandra.
