**NoSQL (Not Only SQL)**

**History --** Carl Strozzi 1998, named his File-based database as NoSQL.

**What is NoSQL?**

Set of **features** for NoSQL database:

1) No-relational Data Model

2) Runs well on clusters

3) Mostly open source

4) Its schema-less

5) Build for new generation web applications

**Why to use NoSQL?**

Evolution of social media like: Facebook, twitter etc which has **user-driven** content has increased and it thereby increased the volume and type of data that is produced, managed, analyzed and achieved. In addition new source of data such as sensors, GPS, trackers etc generate huge volume of data on regular basis.

This huge volume (big data) has introduced new challenges in storing, managing, analyzing and archiving data. In addition data has become semi-structured and sparse.

**What is Semi-structured and unstructured data?**

**Unstructured** data is data that cannot be contained in a row-column database and doesn’t have an associated data model.

Ex: **text** of email message, photos, video and audio files, text files, social media content, satellite imagery, presentations, PDFs, open-ended survey responses, websites and call center transcripts/recordings.

**Semi-structure** data has combination of both kind of structured as well as semi-structured data.

Ex: Email messages are a good example. While the actual content is unstructured, it does contain structured data such as name and email address of sender and recipient, time sent, etc.

**Differences:**

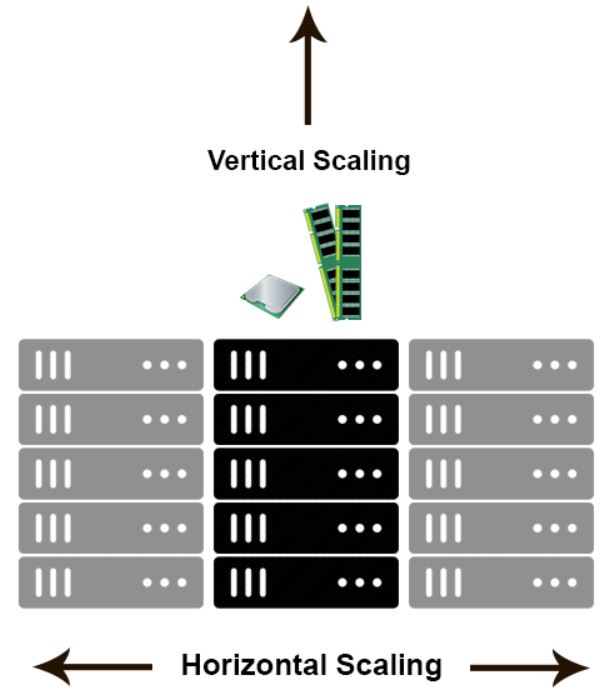
|  |  |
| --- | --- |
| Relational (RDBMS) | NoSQL database System |
|  |  |
| * Data stored in relational model, with rows and columns | * Data stored in host of different database-   Each with different data storage models |
| * Follows fixed schema | * Follows dynamic schema. |
| * Supports vertical scaling | * Supports horizontal scaling |
| * ACID compliant | * Is not ACID compliant |
| * Single point of failure | * No single point of failure(RESILIENT) |
| * Supports structured data | * Supports structured, unstructured and semi-structured data. |
|  |  |

**Vertical and horizontal scaling:**

When to design a system scaling plays a very important role.

**When to scale:**

* Let's start with the need for scaling that is increasing resources so that your system can now handle more requests than it earlier could.
* When you realize your system is getting slow and is unable to handle the current number of requests, you need to scale the system.



**Ways of Scaling:**

1. **Vertical Scaling**

* Either you increase the resources in the server which you are using currently, i.e., increase the amount of RAM, CPU, and GPU etc. This is known as vertical scaling (***Buy bigger Machines***).
* Vertical scaling is typically costly. It does not make the system **fault tolerant,** i.e. if you are scaling application running with single server, if that server goes down, your system will go down.
* Also the amount of threads remains the same in vertical scaling. Vertical scaling may require your system to go down for a moment when process takes place. Increasing resources on a server requires a restart and put your system down.

1. **Horizontal Scaling**

* Another solution to this problem is increasing the amount of servers present in the system. This solution is highly used in the tech industry (***Buy More Machines***).
* This will eventually decrease the request per second rate in each server. If you need to scale the system, just add another server, and you are done.
* You would not be required to restart the system.
* Number of threads in each system decreases leading to high throughput.
* To segregate the requests, equally to each of the application server, you need to add **load balancer** which would act as reverse proxy to the web servers.
* This whole system can be called as a single cluster. Your system may contain a large number of requests which would require more amount of clusters like this.

**Problem with RDBMS in FTCS**:

1. Storing huge MDL file as BLOB into SQL tables. As MDL is semi-structured data so storing it in BLOB and traversing in MDL is really difficult. I have seen that in **CLOBUPDATE**.
2. Trigger

This made the evolution of different categories of NoSQL database:

1) Column based -- CASSANDRA, HBASE

2) Graph based -- Neo4J, ALLEGROGRAPH

3) Document based -- CouchDB,MONGODB,COSMOS DB

4) Key value based database -- AMAZON DYNAMODB, RIAK KV, REDIS

**CASSANDRA DATABASE SYSTEM**

**High Scalable** – It facilitates you to add more hardware (nodes) when requirement increases.

**Flexible Data Store**- Supports all forms of data structure and allows us to change whenever required.

**Fault tolerant** – Allows replication of same data across multiple nodes.

**NoSQL database** and supports easy replication (Simplicity of design).

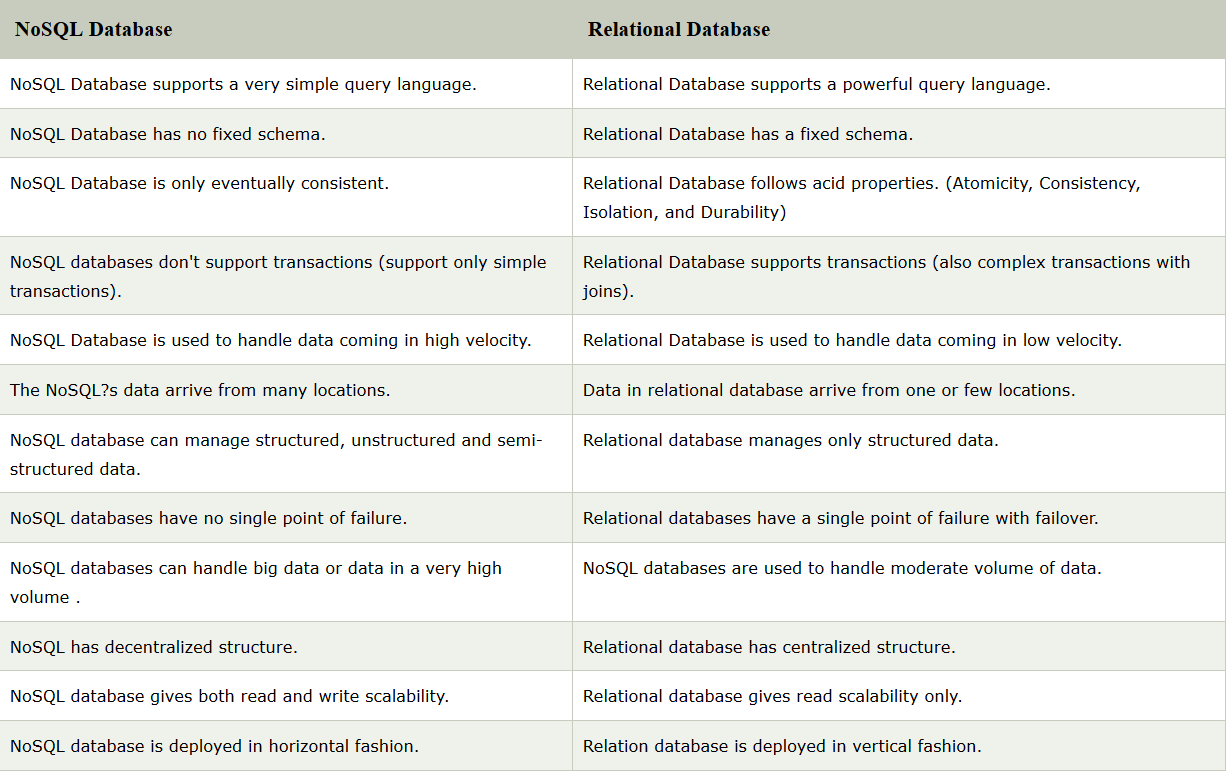
**High Availability** and **horizontal scaling**.

**Why Popular?**

* Open source and distributed storage system.
* High availability with no single point of failure.

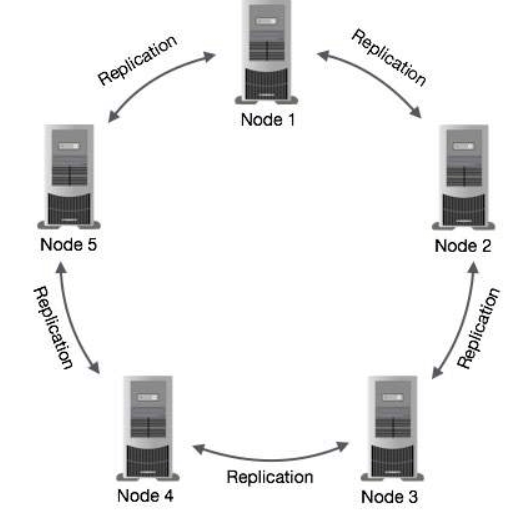
**HISTORY**

* Developed at Facebook by two Indians in July 2008.
* It was accepted by Apache Incubator in March 2009.
* The latest version of Apache Cassandra is 3.2.1.



**ARCHITECTURE**

Cassandra was designed to handle big data workloads across multiple nodes without single point of failure



* Each node is independent and at the same time interconnected to other nodes.
* 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.

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.

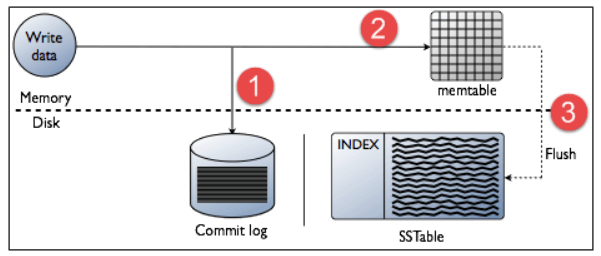
After returning the most recent value, Cassandra performs a **read repair** in the background to update the stale values.

## 

## Components of Cassandra

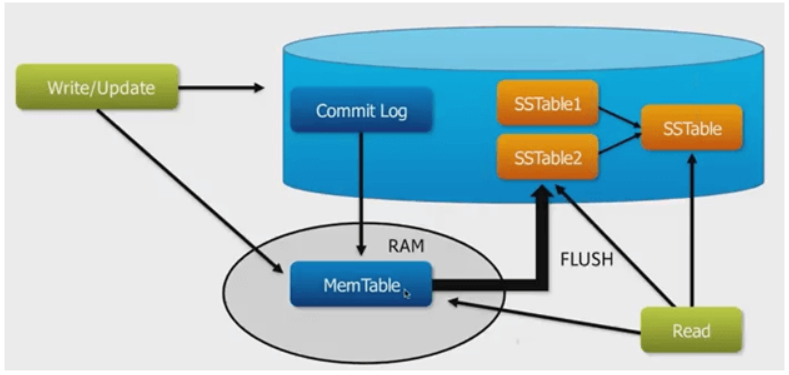
* **Node**: A Cassandra node is a place where data is stored.
* **Data center**: Data center is a collection of related nodes.
* **Cluster**: A cluster is a component which contains one or more data centers.
* **Commit log**: In Cassandra, the commit log is a crash-recovery mechanism. Every write operation 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.
* **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.

**Write Operation**



* 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 Operation**



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** (node) 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.

**USE CASES / APPLICATIONS OF CASSANDRA**

**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 Catalogs 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 Query Language

Cassandra Query Language (**CQL**) is used to access data 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 Automatic Data Expiration

Cassandra provides functionality by which data can be automatically expired.

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

**CASSANDRA DATA MODEL**

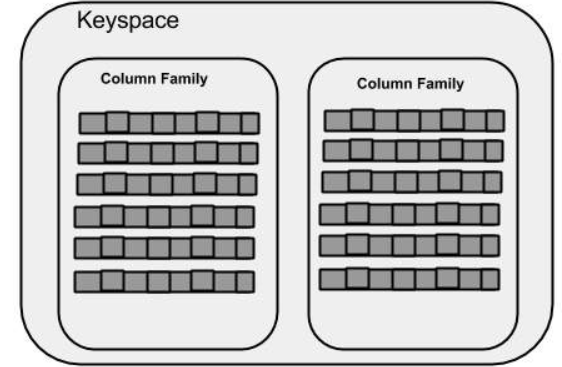
## Cluster

* 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**

* Outermost container for data, which defines the nodes for the data.
* **Replication factor:** It specifies the number of machine 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 (datacenter-shared strategy)

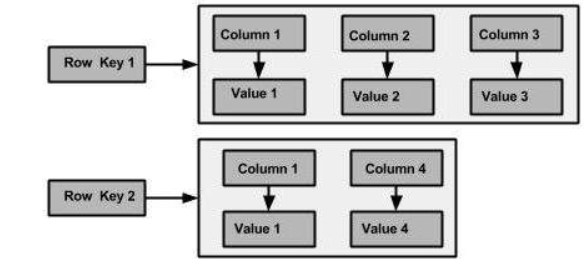


CREATE KEYSPACE Keyspace name

WITH replication = {'class': 'SimpleStrategy', 'replication\_factor' : 3};

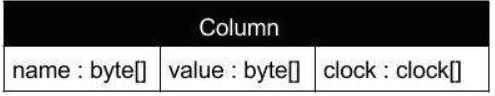
**Column Family**

* 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, although the **column families are defined, the columns are not**. You can freely add any column to any column family at any time



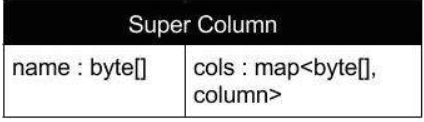
**Column**

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.



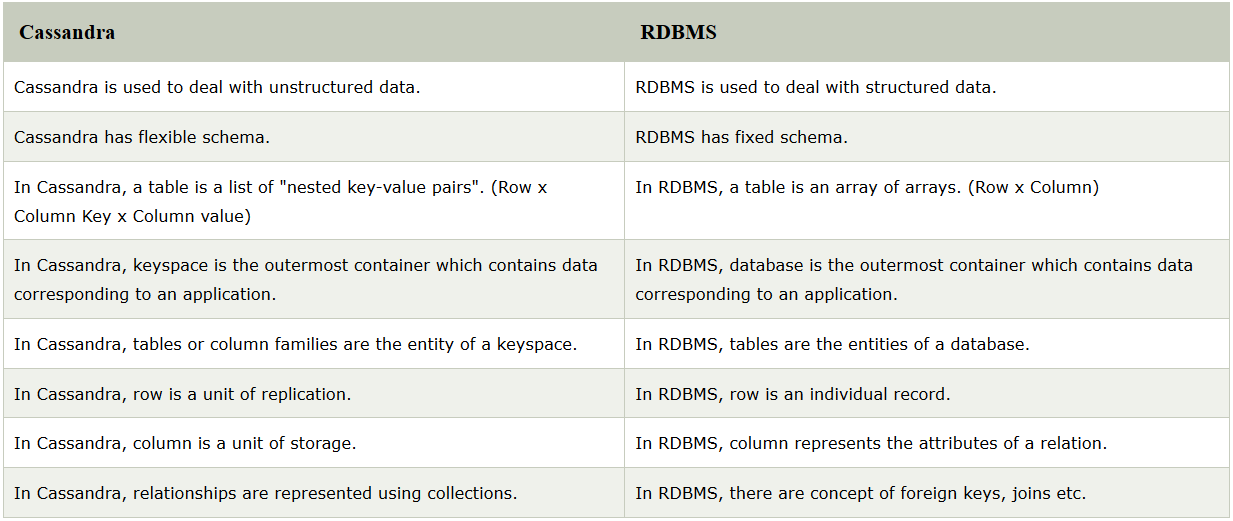
**SuperColumn**

* 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 column family, and a super column can be helpful here.



## 

## Data Models of Cassandra and RDBMS



REFERENCES::

https://www.youtube.com/watch?v=9Y-Q7BTqOw4&list=PLiLpmqwkwkCtUtXioKVyB0jpcxro0qQAh&index=6&t=0s